Fundamentals of Machine Learning and Analyzing Data with Python

A Practical Approach

ความรู้พื้นฐานทางด้านการเรียนรู้เครื่องจักร และการวิเคราะห์ข้อมูลด้วยโปรแกรมภาษาไพธอน

ด้วยวิธีฝึกปฏิบัติ

Olarik Surinta

Fundamentals of Machine Learning and Analyzing Data with Python

A Practical Approach

ความรู้พื้นฐานทางด้านการเรียนรู้เครื่องจักร และการวิเคราะห์ข้อมูลด้วยโปรแกรมภาษาไพธอน

ด้วยวิธีฝึกปฏิบัติ

Fundamentals of Machine Learning and Analyzing Data with Python

A Practical Approach

ความรู้พื้นฐานทางด้านการเรียนรู้เครื่องจักร และการวิเคราะห์ข้อมูลด้วยโปรแกรมภาษาไพธอน

ด้วยวิธีฝึกปฏิบัติ

Olarik Surinta

Department of Information Technology
Faculty of Informatics, Mahasarakham University
Maha Sarakham, Thailand

คำนำ

เทคโนโลยีสารสนเทศสมัยใหม่ทำให้โลกเกิดการปฏิวัติทำให้มนุษย์พัฒนานวัตกรรม ไม่ว่าจะเป็นสิ่งประดิษฐ์ที่สามารถจับต้องได้ ไปจนซอฟต์แวร์ที่สามารถใช้กันอย่างแพร่หลายผ่านระบบเครือข่ายอินเทอร์เน็ต ทำให้เกิดระบบที่ทำงานผ่านโลกออนไลน์มีอัตราการเจริญเติบโตสูงขึ้นอย่างก้าวกระโดดในทุกปี อีกทั้งยังเกิดข้อมูลมหาศาลที่อยู่บนโลกออนไลน์ ดังนั้น จึงทำให้สามารถนำข้อมูลเหล่านั้นมาวิเคราะห์ และประมวลผล (Data Analytics) เพื่อนำแบบจำลองไปใช้เป็นตัวช่วยในการวิเคราะห์และพัฒนาธุรกิจ ทำให้สามารถมีสารสนเทศที่จะใช้สำหรับแข่งขันกับคู่แข่งทางธุรกิจ การวิเคราะห์ข้อมูลนั้นต้องอาศัยความรู้ทางด้านต่าง ๆ มากมาย เช่น เหมืองข้อมูล (Data Mining) การเรียนรู้เครื่องจักร (Machine Learning) และวิทยาการข้อมูล (Data Science) ซึ่งเป็นแนวโน้ม (Trend) ใหม่ของโลกในยุคปัจจุบัน ซึ่งความรู้เหล่านี้มีส่วนช่วยให้สามารถเลือกใช้อัลกอริธึ่มได้ถูกต้อง ตรงกับปัญหา และตอบโจทย์ทางธุรกิจมากที่สุด ในหนังสือเล่มนี้ได้นำเสนอเกี่ยวกับการใช้โปรแกรมภาษาไพธอน (Python) ซึ่งเป็นภาษาที่นิยมใช้งานกันในกลุ่มนักวิเคราะห์ข้อมูล (Data Analyst) ซึ่งมีไลบรารี่ (Library) และเครื่องมือ (Tool) ให้เลือกใช้มากมาย

สารบัญ

บทที่ 1 Jupyter Notebook.....1
วิธีการติดตั้งโปรแกรม Jupyter Notebook.....1
วิธีการเรียกใช้โปรแกรม Jupyter Notebook.....2
ไฟล์นามสกุล (File Extension) ของโปรแกรม Jupyter.....3

บทที่ 2 การโหลดข้อมูล (Loading Data).....5
Iris Dataset.....5
ลักษณะของชุดข้อมูล Iris.....7
MNIST Dataset.....9
การ Visualization เพื่อดูรูปภาพตัวเลข.....13
ข้อมูลที่นำไปใช้ใน scikit-learn.....14
โหลดชุดข้อมูล MNIST ด้วย Scipy.....15
การ Visualization ชุดข้อมูล MNIST.....18

บทที่ 3 ข้อมูลชุดเรียนรู้และข้อมูลชุดทดสอบ (Training and Test Data).....20
การกำหนดอัตราที่ใช้สำหรับแบ่งข้อมูล.....21
การแสดงข้อมูลในรูปแบบของกราฟ.....21

บทที่ 4 ไลบรารี่ Matplotlib (Matplotlib Library).....25
การบันทึกกราฟ (Figure).....27
การแสดงรูปภาพ (Image Show).....28
วิธีการแสดงรูปภาพโดยใช้อินเทอร์เฟซ (Interface) ที่ต่างกัน.....28
การเพิ่มเส้นตาราง (Grid) ในการพลอต.....30
การกำหนดวิธีการพลอต: Line Color และ Style.....34
การกำหนด Axes Limit.....37
การพลอต Label.....39
การพลอต Legend.....39
การกำหนดการพลอต Legend.....40
การพลอตแบบ Scatter.....45
การตกแต่ง Scatter.....49
การพลอต Histograms, Binnings และ Density.....52
การคำนวณค่า Histogram.....54
การพลอตชุดข้อมูล MNSIT ที่อยู่ใน scikit-learn.....55
การพลอตชุดข้อมูล MNIST ในรูปแบบ 2 มิติ โดยใช้วิธี IsoMap.....55

บทที่ 5 ไลบรารี่ Seaborn (Seaborn Library).....57
การแสดงชุดข้อมูล Iris แบบ Visualization.....58
การทำงานร่วมกันระหว่าง seaborn และ scikit-learn.....59

บทที่ 6 การวิเคราะห์การถดถอยเชิงเส้น (Linear Regression).....62
การจำลองชุดข้อมูลเพื่อใช้ในการคำนวณ Linear Regression.....63
การเรียกใช้โมดูล LinearRegression.....63
จัดการข้อมูลเพื่อใช้ในการคำนวณ.....64
การเรียนรู้เพื่อสร้างโมเดล (Train the model).....64

การพยากรณ์ผลลัพธ์จากข้อมูลใหม่ (Predict Labels for Unknown Data).....65
การพยากรณ์ข้อมูล Diabetes ด้วย Linear Regression.....66
การพยากรณ์ข้อมูล Housing ด้วย Linear Regression.....71

บทที่ 7 ตัวจำแนกแบบไบนารี่ (Binary Classifier).....79
Stochastic Descent.....81
สร้างโมเดล Stochastic Descent.....82
การพยากรณ์ด้วยโมเดล Stochastic Descent.....82
การวัดประสิทธิภาพ (Performance Measurement).....84
การประเมินประสิทธิภาพของอัลกอริธึมด้วย Confusion Matrix.....84
การนำโมเดลไปทดสอบกับข้อมูลชุดทดสอบ.....87

บทที่ 8 การคำนวณเพื่อนบ้านใกล้ที่สุด k ตำแหน่ง (K-Nearest Neighbors).....89
การสร้างโมเดลของ KNN.....90
การพยากรณ์โดยใช้โมเดลของ KNN.....91
การทดสอบประสิทธิภาพของโมเดล KNN.....92
การแสดงผลการทดลองด้วย Confusion Matrix.....92
การใช้งานอัลกอริธึม KNN กับข้อมูลโรคเบาหวาน (Diabetes Dataset).....95
สร้างโมเดล KNN ด้วยค่า n_neighbor ที่ได้จากการทดลอง.....97
คำสั่ง Pandas Crosstab.....98
แสดงผลลัพธ์จากการทดลองด้วย Classification Report.....98
การทดสอบค่าพารามิเตอร์ (Hyperparameter Tuning) ด้วยวิธี Grid Search.....99
การจัดหมวดหมู่ชุดข้อมูล MNIST ด้วยอัลกอริธึม KNN.....100
KNN Classifier.....101

บทที่ 9 การจัดหมวดหมู่ข้อมูลด้วย Naive Bayes (Naive Bayes Classification).....105
การเตรียมข้อมูลเพื่อใช้ในการเรียนรู้.....107
สร้างโมเดล Naive Bayes.....108
พยากรณ์ข้อมูลด้วยโมเดล Naive Bayes และแสดงประสิทธิภาพของโมเดล.....108

บทที่ 10 การวิเคราะห์องค์ประกอบหลัก (Principal Component Analysis).....111
การสร้างโมเดล PCA.....111
การเพิ่มข้อมูลจากตัวแปรเข้าไปเก็บเพิ่มใน DataFrame.....112
สร้างโมเดล Naive Bayes ด้วยคุณลักษณะพิเศษที่ได้จาก PCA.....115
การพยากรณ์คุณลักษณะพิเศษที่ได้จาก PCA ด้วยอัลกอริธึม Naive Bayes และประสิทธิภาพจากการพยากรณ์ 117

บทที่ 11 การจัดกลุ่มด้วยอัลกอริธึม K-Means (K-Means Clustering).....119
จำลองข้อมูลเพื่อใช้ในอัลกอริธึม K-Means.....120
สร้างโมเดลของอัลกอริธึม K-Means.....121
การพยากรณ์ด้วยอัลกอริธึม K-Means.....121
สร้างข้อมูลใหม่เพื่อทดสอบการแบ่งกลุ่มด้วยอัลกอริธึม K-Means.....123
พยากรณ์ข้อมูลที่สร้างขึ้นมาใหม่ด้วยอัลกอริธึม K-Means.....124

บทที่ 12 การรู้จำใบหน้า (Face Recognition).....127
สร้างโมเดลของอัลกอริธึม SVM.....128
การวัดประสิทธิภาพของการรู้จำใบหน้า (Classification Report).....130
การแสดงความถูกต้องของการพยากรณ์ด้วย Confusion Matrix.....131
อัตราความถูกต้อง (Accuracy Result) ของการพยากรณ์รูปภาพใบหน้า.....131

บทที่ 13 การรู้จำตัวอักษร (Character Recognition).....133
สร้างโมเดลของอัลกอริธึม MLP.....136
การวัดประสิทธิภาพของการเรียนรู้.....136
การพยากรณ์และวัดประสิทธิภาพของการรู้จำ.....137
การ Visualization รูปภาพตัวเลข และแสดงผลการพยากรณ์.....138

สามารถดาวน์โหลดตัวอย่างโปรแกรม และบทเรียนนี้ในรูปแบบของ Jupyter Notebook ได้จากเว็บไซต์ github

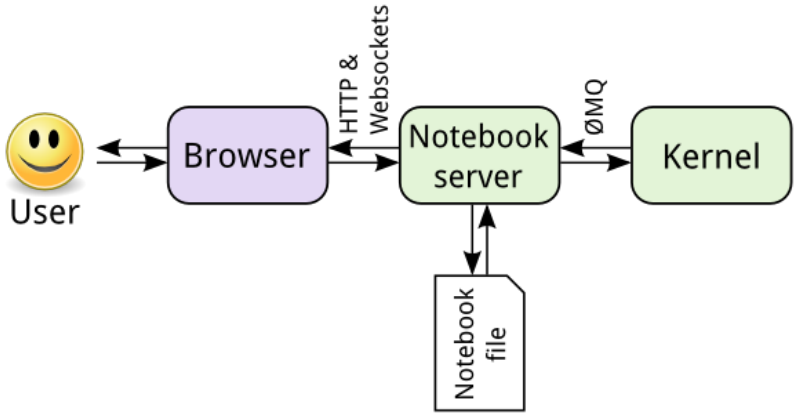
https://github.com/mrolarik/basic-machine-learning-using-scikit-learn

สารบัญภาพประกอบ

ภาพประกอบที่ 1: ลักษณะการทำงานของโปรแกรม Jupyter Notebook.....1
ภาพประกอบที่ 2: ตัวอย่างการเรียกใช้ jupyter notebook ผ่านหน้าจอ Terminal.....2
ภาพประกอบที่ 3: ตัวอย่างโปรแกรม Jupyter.....3
ภาพประกอบที่ 4: ตัวอย่างโครงสร้างการเก็บข้อมูลของ Jupyter โดยจัดเก็บในรูปแบบ JSON.....3
ภาพประกอบที่ 5: ชุดข้อมูลดอกไม้ Iris ประกอบด้วยดอกไม้ 3 สายพันธุ์ ได้แก่ Versicolor, Setosa และ Virginica.....5
ภาพประกอบที่ 6: ตัวอย่างการเก็บข้อมูลของชุดข้อมูล iris.....8
ภาพประกอบที่ 7: ตัวอย่างของข้อมูลชุด MNSIT.....10
ภาพประกอบที่ 8: การ Visualization ชุดข้อมูล MNIST พร้อมทั้งแสดง label ของแต่ละตัวเลข.....19
ภาพประกอบที่ 9: แสดงสมการถดถอยเชิงเส้น Linear Regression.....62
ภาพประกอบที่ 10: เส้น Hyperplane ที่ได้จากการคำนวณด้วยวิธี Linear Regression.....66
ภาพประกอบที่ 11: แสดงการพลอตค่า target จริง (Actual Target) และค่า target ที่ได้จากการพยากรณ์ (Predicted Target).....70
ภาพประกอบที่ 12: ตัวอย่างข้อมูลที่จัดเก็บในรูปแบบของ pandas DataFrame.....73
ภาพประกอบที่ 13: แสดงเส้น Hyperplane ที่ใช้แบ่งข้อมูลออกเป็นสองส่วน.....79
ภาพประกอบที่ 14: แสดงลักษณะการทำงานของอัลกอริธึม KNN.....89
ภาพประกอบที่ 15: กราฟแสดงประสิทธิภาพของอัลกอริธึม KNN เมื่อเปลี่ยนค่าพารามิเตอร์ n_neighbors.....97
ภาพประกอบที่ 16: ตัวอย่างการจัดกลุ่มด้วยวิธี K-Means โดยแสดงให้ดูทีละขั้นตอน.....119
ภาพประกอบที่ 17: ตัวอย่างใบหน้าจากชุดข้อมูล lfw ที่นำมาใช้ในการรู้จำใบหน้า.....128
ภาพประกอบที่ 18: ผลลัพธ์ที่ได้จากการพยากรณ์ใบหน้าด้วยวิธี PCA และ SVM.....130
ภาพประกอบที่ 19: ตัวอย่างโครงสร้างของ Multi-Layer Perceptron (MLP).....133
ภาพประกอบที่ 20: เปรียบเทียบระหว่างค่าที่แท้จริง และค่าที่ได้จากการพยากรณ์.....139

บทที่ 1
Jupyter Notebook

Jupyter หรือ Jupyter Notebook เป็นเคอร์นอล (Kernel) ของโปรแกรมภาษา Python ที่ทำให้สามารถคำนวณและติดต่อสื่อสารกับส่วนติดต่อกับผู้ใช้งาน (Frontend) ได้สะดวก และเป็นการทำงานผ่านเว็บบราวเซอร์ (Browser)



ภาพประกอบที่ 1: ลักษณะการทำงานของโปรแกรม Jupyter Notebook

วิธีการติดตั้งโปรแกรม Jupyter Notebook

การติดตั้งโปรแกรม Jupyter Notebook สามารถติดตั้งผ่านเทอร์มินัล (Terminal) โดยใช้คำสั่ง

```
$ pip install jupyter
```

ตัวอย่างข้างต้นเป็นการติดตั้งสำหรับ Python เวอร์ชัน 2 แต่สำหรับ Python เวอร์ชัน 3 จะต้องใช้คำสั่ง ดังนี้

```
$ pip3 install jupyter
```

วิธีการเรียกใช้โปรแกรม Jupyter Notebook

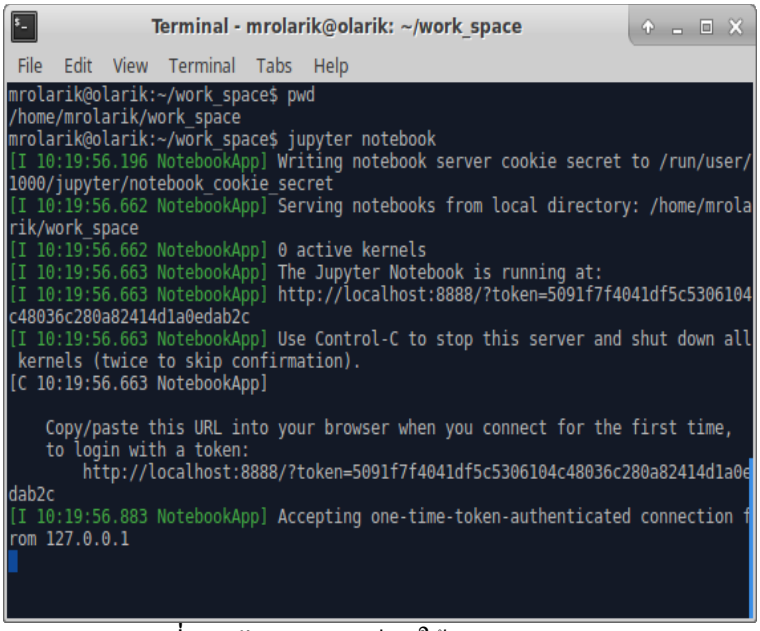
เมื่อติดตั้งโปรแกรม Jupyter เสร็จเรียบร้อย สามารถเรียกใช้โปรแกรม Jupyter โดยพิมพ์คำสั่งผ่าน Terminal ดังนี้

```
$ jupyter notebook
```

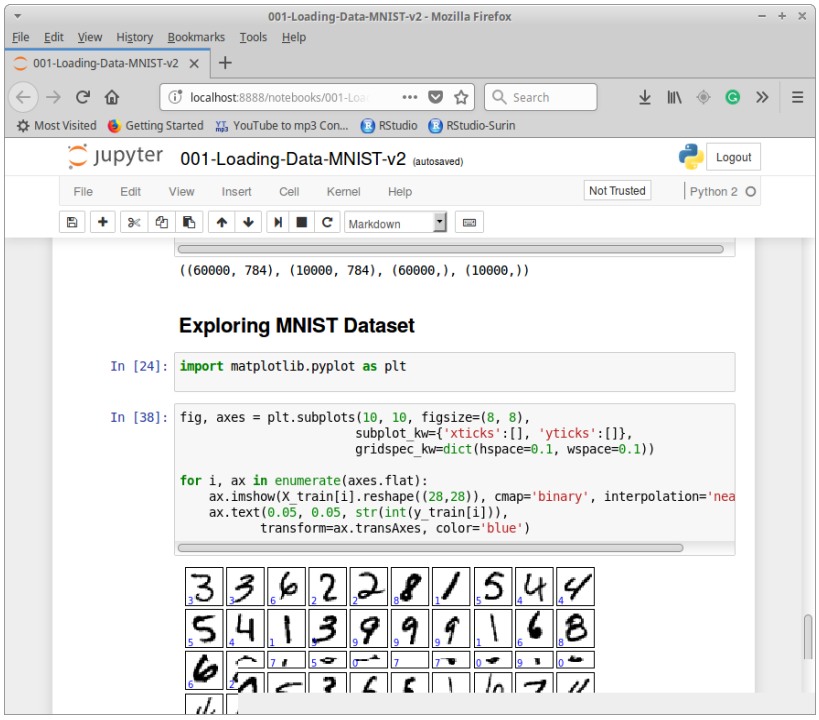
เมื่อเรียกใช้คำสั่ง $ jupyter notebook จากนั้นโปรแกรมจะทำการจำลองเป็นเซิร์ฟเวอร์ (Server) เพื่อให้สามารถทำงานผ่านเว็บบราวเซอร์ โดยจะเปิดเว็บบราวเซอร์ ณ ตำแหน่งที่อยู่ปัจจุบัน ดังนั้น หากต้องการที่จะให้โปรแกรม Jupyter ทำงานในโฟลเดอร์ที่ต้องการ ให้เปิด Terminal และไปยังตำแหน่งที่ต้องการ จากนั้นจึงจะเรียกใช้คำสั่ง $ jupyter notebook (ดังภาพประกอบที่ 2) เช่น

```
$ cd /home/user/work_space
$ jupyter notebook
```

จากตัวอย่าง เมื่อเรียกใช้โปรแกรม jupyter โปรแกรมจะเปิดเว็บบราวเซอร์และทำงาน ณ ตำแหน่ง /home/user/work_space โดย user เป็นโฟลเดอร์ที่เปลี่ยนไปตามชื่อผู้ใช้งาน โดย jupyter notebook จะจำลองตัวเป็นเซิร์ฟเวอร์เพื่อให้สามารถเปิดและทำงานผ่านเว็บบราวเซอร์ (ดังภาพประกอบที่ 3)



ภาพประกอบที่ 2: ตัวอย่างการเรียกใช้ jupyter notebook ผ่านหน้าจอ Terminal



ภาพประกอบที่ 3: ตัวอย่างโปรแกรม Jupyter

ไฟล์นามสกุล (File Extension) ของโปรแกรม Jupyter

โปรแกรม Jupyter จะมีนามสกุล (Extension) เป็น .ipynb ซึ่งจะแตกต่างจากโปรแกรมภาษา Python ทั่วไปที่จะใช้นามสกุล .py โดยโปรแกรม Jupyter จะเก็บข้อมูลทั้งหมดในรูปแบบของ JSON แสดงดังภาพประกอบที่ 4

```
{
 "cells": [
  {
   "attachments": {},
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# Jupyter\n",
    "\n",
    "Jupyter Notebook เป็น Kernel ของโปรแกรม Python ที่ทำให้สามารถคำนวณ และติดต่อสื่อสาร
กับ frontend interface โดยเป็นการทำงานผ่านเว็บบราวเซอร์\n",
    "\n",
    "![alt text]
(http://jupyter.readthedocs.io/en/latest/_images/notebook_components.png \
"Kernel\")"
   ]
  },
```

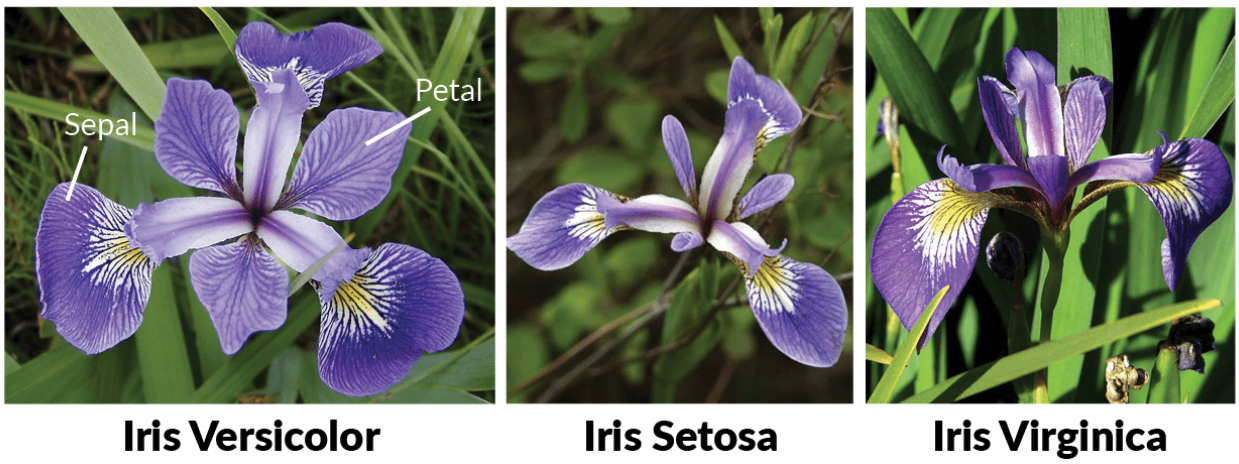
ภาพประกอบที่ 4: ตัวอย่างโครงสร้างการเก็บข้อมูลของ Jupyter โดยจัดเก็บในรูปแบบ JSON

บทที่ 2
การโหลดข้อมูล (Loading Data)

ในบทนี้ นำเสนอชุดข้อมูล (Dataset) ที่ใช้กันอย่างแพร่หลายในงานด้านการเรียนรู้เครื่องจักร (Machine Learning) โดยข้อมูลดังกล่าว ประกอบด้วย Iris และ MNSIT dataset ซึ่งได้นำเสนอลักษณะของข้อมูล และวิธีการเรียกใช้ข้อมูล

Iris Dataset

ชุดข้อมูลดอกไม้ Iris (Iris Dataset) เป็นชุดข้อมูลพื้นฐานที่ใช้อย่างแพร่หลายทั้งทางด้านการเรียนรู้เครื่องจักร และทางสถิติ (Statistics) หากติดตั้งโปรแกรม scikit-learn จะสามารถเรียกใช้ชุดข้อมูล Iris ได้ทันที ตัวอย่างรูปภาพดอกไม้ Iris แสดงดังภาพประกอบที่ 5



ภาพประกอบที่ 5: ชุดข้อมูลดอกไม้ Iris ประกอบด้วยดอกไม้ 3 สายพันธุ์ ได้แก่ Versicolor, Setosa และ Virginica

หากติดตั้งโปรแกรม scikit-learn สามารถเรียกใช้ชุดข้อมูล Iris ซึ่งเป็นชุดข้อมูลดอกไม้ ทำได้ดังต่อไปนี้

วิธีที่ 1

```
from sklearn import datasets
iris_dataset = datasets.load_iris()
```

วิธีที่ 2

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

จากตัวอย่างข้างต้นฟังก์ชัน (Function) load_iris() เป็นฟังก์ชันที่อยู่ในโมดูล (Module) สามารถเรียกใช้ได้เมื่อติดตั้งและอิมพอร์ต (Import) scikit-learn มาใช้งาน เมื่อเรียกใช้ฟังก์ชัน load_iris() จะคืนค่า (Return) ออกมาเป็น Bunch object ซึ่งเป็นโครงสร้างแบบ dictionary ในภาษา Python ประกอบไปด้วย keys และ values การใช้งานสามารถทำได้ดังต่อไปนี้

```
print("Keys of iris_dataset: \
     \n{}".format(iris_dataset.keys()))
```

```
Keys of iris_dataset:

['target_names', 'data', 'target', 'DESCR', 'feature_names']
```

จากตัวอย่าง DESCR คือ key ที่เก็บค่า (Value) โดยค่านั้นคือรายละเอียดของชุดข้อมูล iris แสดงดังต่ออย่างต่อไปนี้

```
print(iris_dataset['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
```

ลักษณะของชุดข้อมูล Iris

ชุดข้อมูล iris มีเอาท์พุท หรือ target อยู่ 3 กลุ่ม ประกอบด้วย versicolor, setosa และ virginica สามารถเรียกดูโดยใช้คำสั่ง

```
print("Target names: \
     \n{}".format(iris_dataset['target_names']))
```

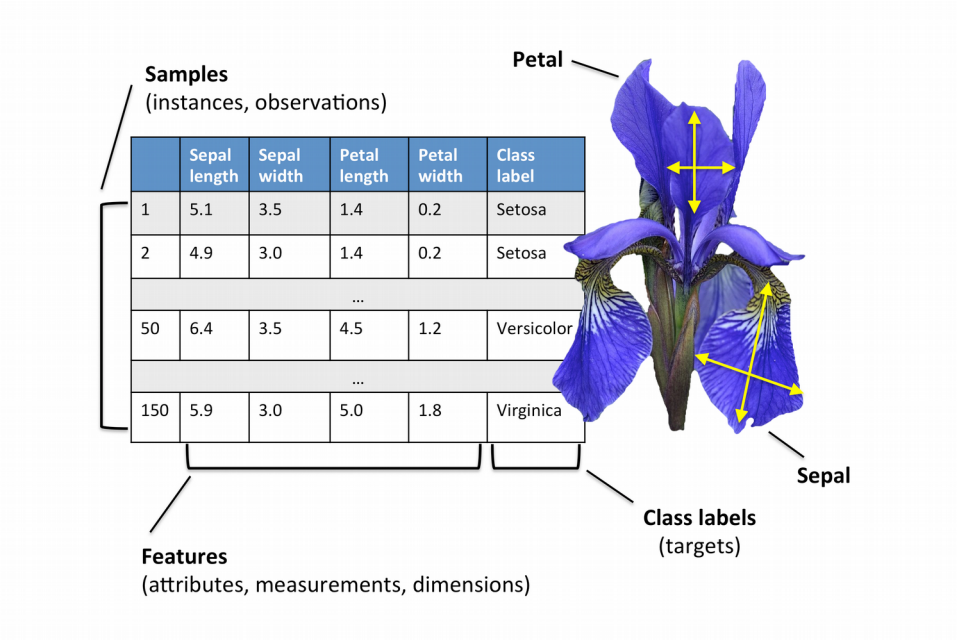
```
Target names:

['setosa' 'versicolor' 'virginica']
```

หากต้องการแสดงรายชื่อของ feature หรือ attribute ของชุดข้อมูล iris สามารถทำได้โดย

```
print("Feature names:  \
     \n{}".format(iris_dataset['feature_names']))
```

```
Feature names:

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

ในแต่ละ feature ที่ปรากฏในชุดข้อมูล iris คือการวัดความกว้างและความยาวของกลีบใบที่เรียกว่า Sepal และ Petal โดยมีหน่วยเป็นเซ็นติเมตร (cm.) ตัวอย่างของการเก็บข้อมูลแสดงดังภาพประกอบ 6



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

ภาพประกอบที่ 6: ตัวอย่างการเก็บข้อมูลของชุดข้อมูล iris

การเรียกดูข้อมูลสามารถทำได้โดยเรียกใช้ Key ที่ชื่อ data โดยตัวอย่างดังต่อไปนี้ แสดงให้เห็นถึงการเรียกใช้ข้อมูลจำนวน 10 ชุด

```
print("First ten rows of data:\n \
     {}".format(iris_dataset['data'][0:10]))
```

```
First ten rows of data:

 [[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]]
```

หากต้องการเรียกดูข้อมูล class สามารถทำได้โดยเรียกใช้ key ที่ชื่อ target ดังตัวอย่างต่อไปนี้

```
# ตัวอย่างผลลัพธ์ หรือ class ของข้อมูล
print("Target:\n{}".format(iris_dataset['target']))
```

```
Target:

[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 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]
```

จากตัวอย่างข้างต้น

0 หมายถึง iris setosa
1 หมายถึง iris versicolor
2 หมายถึง virginica

หากต้องการเรียกดูขนาดของ target หรือจำนวนของข้อมูลสามารถทำได้โดย

```
# แสดงจำนวนของผลลัพธ์
print("shape of target: \
     {}".format(iris_dataset['target'].shape))
```

```
shape of target: (150,)
```

หากต้องการเรียกดูประเภทของข้อมูลที่ใช้จัดเก็บสามารถทำได้โดย

```
# ประเภทข้อมูลของ target
print("Type of target: \
     {}".format(type(iris_dataset['target'])))
```

```
Type of target: <type 'numpy.ndarray'>
```

ประเภทของข้อมูล (Data Type) ที่ใช้จัดเก็บคือ numpy.ndarray ซึ่ง numpy เป็นไลบรารี่ที่ใช้จัดการข้อมูลประเภทตัวเลขมีการจัดเก็บแบบอาร์เรย์ (Array)

MNIST Dataset

ข้อมูลชุด MNIST เป็นชุดข้อมูลลายมือตัวเลขอารบิก (ตัวเลข 0-9) โดยรูปภาพมีขนาด 28x28 พิกเซล และมีข้อมูลในชุดเรียนรู้จำนวน 60,000 รูปภาพ และข้อมูลชุดทดสอบจำนวน 10,000 รูปภาพ สำหรับ scikit-learn นั้น MNIST เป็นชุดข้อมูลตัวอย่างที่กำหนดให้รูปภาพตัวอักษรลายมือตัวเลขอารบิกในแต่ละรูปมีขนาด 8x8 พิกเซล และมีจำนวน 5,620 ชุดเท่านั้น ตัวอย่างข้อมูลชุด MNIST แสดงดังภาพประกอบที่ 7

ภาพประกอบที่ 7: ตัวอย่างของข้อมูลชุด MNSIT

การเรียกใช้ชุดข้อมูล MNIST ที่มาพร้อมกับ scikit-learn สามารถทำได้ดังต่อไปนี้

```
from sklearn import datasets
digits = datasets.load_digits()
```

ดังนั้น หากต้องการดูรายชื่อของ key ของชุดข้อมูล MNSIT สามารถทำได้โดย

```
print("Keys of digits: \
     \n{}".format(digits.keys()))
```

```
Keys of digits:
['images', 'data', 'target_names', 'DESCR', 'target']
```

หากต้องการดูรายละเอียดของชุดข้อมูล MNIST ทำได้โดย

```
print(digits['DESCR'])
```

```
Optical Recognition of Handwritten Digits Data Set
===================================================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 5620
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
```

```
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.
```

หากต้องการดูขนาดของชุดข้อมูล MNIST สามารถใช้คำสั่ง shape ซึ่งเป็นเมธอด (Method) หนึ่งที่ใช้ร่วมกับไลบรารี่ numpy

```
print("size of the MNIST dataset", digits.images.shape)
```

```
('size of the MNIST dataset', (1797, 8, 8))
```

สามารถเรียกดูข้อมูลชื่อของ target ได้ดังนี้

```
print("target name", digits.target_names)
```

```
('target name', array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]))
```

รูปภาพตัวเลขอารบิกที่จัดเก็บมีขนาด 8x8 พิกเซล หากต้องการตรวจสอบข้อมูลของแต่ละตัวเลขสามารถเรียกดูได้ดังนี้

```
print("Shape of each image", \
     digits.images[0].shape)

digits.images[0]
```

```
('Shape of each image', (8, 8))

array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

คำสั่ง digits.images[0].shape ใช้เพื่อดูขนาดของตัวเลขในตำแหน่งที่ 0 และคำสั่ง digits.images[0] เป็นการแสดงข้อมูลทั้งหมดของตัวเลขในตำแหน่งที่ 0 ดังนั้น หากต้องการดูรายละเอียดของข้อมูลมากกว่า 1 ชุด สามารถใช้คำสั่งดังต่อไปนี้

```
#แสดงตัวอย่างข้อมูลจำนวน 2 ชุด ตั้งแต่ชุดที่ 0 ถึง 2
print("Sample image 0 to 2")
digits.images[0:2]
```

```
Sample image 0 to 2

array([[[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
        [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
        [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
        [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
        [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
        [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
        [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
        [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]],

       [[ 0.,  0.,  0.,  4., 15., 12.,  0.,  0.],
        [ 0.,  0.,  3., 16., 15., 14.,  0.,  0.],
        [ 0.,  0.,  8., 13.,  8., 16.,  0.,  0.],
        [ 0.,  0.,  1.,  6., 15., 11.,  0.,  0.],
        [ 0.,  1.,  8., 13., 15.,  1.,  0.,  0.],
        [ 0.,  9., 16., 16.,  5.,  0.,  0.,  0.],
        [ 0.,  3., 13., 16., 16., 11.,  5.,  0.],
        [ 0.,  0.,  0.,  3., 11., 16.,  9.,  0.]]])
```

การดูข้อมูล target ทั้งหมดสามารถทำได้โดยใช้คำสั่ง

```
print("target (y / label / class) from 0 to 20")
digits.target[:20]
```

```
target (y / label / class) from 0 to 20

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

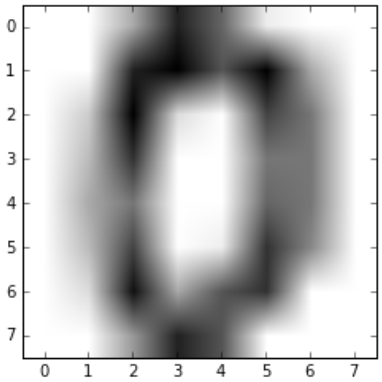
จากตัวอย่างข้างต้น คำสั่ง digits.target[:20] หมายถึงเรียกดูข้อมูลตั้งแต่ตำแหน่งที่ 0 ถึงตำแหน่งที่ 20

การ Visualization เพื่อดูรูปภาพตัวเลข

การแสดงข้อมูล (Visualization) สามารถทำได้หลากหลายวิธี เช่น ใช้ฟังก์ชันจาก scikit-image, matplotlib และ pylab การแสดงข้อมูลสามารถทำได้ดังต่อไปนี้

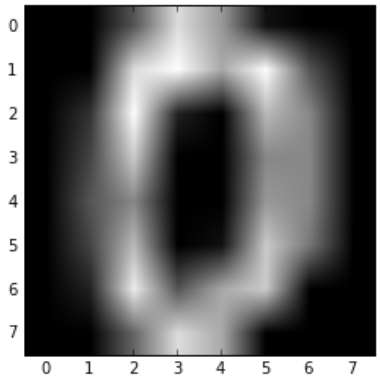
การแสดงข้อมูลด้วย pylab

```
import pylab
pylab.imshow(digits.images[0], cmap=pylab.cm.gray_r)
pylab.show()
```

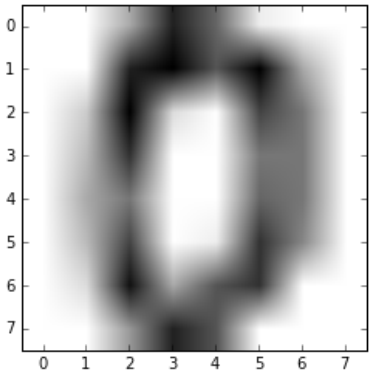


การแสดงข้อมูลด้วย matplotlib

```
import matplotlib.pyplot as plt
plt.imshow(digits.images[0], cmap=plt.get_cmap('gray'))
plt.show()
```



```
import matplotlib.pyplot as plt
plt.imshow(digits.images[0], cmap=plt.cm.gray_r)
plt.show()
```



ข้อมูลที่นำไปใช้ใน scikit-learn

ข้อมูลที่จะนำไปใช้ใน scikit-learn จะต้องแปลงให้เป็นเว็กเตอร์ (Vector) ใน 1 ตัวเลขของชุดข้อมูล MNIST จัดเก็บแบบ array 2 มิติ จึงต้องแปลงให้อยู่ในรูปแบบของ vector ดังนั้นรูปภาพขนาด 8x8 พิกเซล เมื่อแปลงให้เป็น vector จึงเป็นเว็กเตอร์ขนาด 64 การแปลงให้เป็นเว็กเตอร์ในกรณีนี้ใช้คำสั่ง reshape สามารถทำได้ดังต่อไปนี้

```
data = digits.images.reshape((digits.images.shape[0], -1))
print("size of the data", data.shape)
```

```
('size of the data', (1797, 64))
```

จากตัวอย่างข้างต้น การปรับเปลี่ยนรูปร่างข้อมูลทำได้โดยใช้คำสั่ง reshape จากนั้นสามารถใช้คำสั่ง data.shape เพื่อดูขนาดของข้อมูลหลังจากการปรับเปลี่ยน โดยข้อมูลใหม่จะมีขนาดเป็น (1797, 64) สามารถเรียกดูข้อมูลในแต่ละแถวดังต่อไปนี้

```
print("size of each row", data[0].shape)
data[0]
```

```
('size of each row', (64,))

array([ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0.,  0., 13., 15., 10.,
       15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
       12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
        0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
       10., 12.,  0.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.])
```

โหลดชุดข้อมูล MNIST ด้วย Scipy

เนื่องจากการโหลดชุดข้อมูล MNIST จาก scikit-learn สามารถใช้ได้เพียงชุดข้อมูลตัวอย่างที่มีขนาดของรูปภาพเพียง 8x8 พิกเซล และมีจำนวน 5,620 ชุดข้อมูลเท่านั้น ดังนั้น หากต้องการใช้ชุดข้อมูล MNIST ชุดสมบูรณ์จะต้องใช้ข้อมูลจากแหล่งอื่น สามารถดาวน์โหลด MNIST ได้จากลิงก์ต่อไปนี้

https://github.com/amplab/datascience-sp14/blob/master/lab7/mldata/mnist-original.mat

ไฟล์ที่โหลดจะมีนามสกุล .mat ซึ่งจัดเก็บอยู่ในฟอร์แมต (Format) ของโปรแกรม MATLAB ไฟล์มีขนาด 55.4 MB ซึ่งมีขนาดใหญ่พอสมควร เมื่อดาวน์โหลดไฟล์เสร็จเรียบร้อยสามารถเรียกใช้ ดังคำสั่งดังต่อไปนี้

```
from scipy.io import loadmat
mnist_raw = loadmat("mldata/mnist-original.mat")
mnist = {
    "data": mnist_raw["data"].T,
    "target": mnist_raw["label"][0],
    "COL_NAMES": ["label", "data"],
    "DESCR": "mldata.org dataset: mnist-original",
    }
```

จากคำสั่ง **loadmat("mldata/mnist-original.mat")** จะต้องระบุตำแหน่งของไฟล์ให้ถูกต้อง ในตัวอย่างไฟล์ .mat จัดเก็บอยู่ที่โฟลเดอร์ mldata

สามารถดูรายละเอียดของข้อมูล MNSIT โดยพิมพ์คำสั่ง mnist ดังตัวอย่างต่อไปนี้

```
mnist
```

```
{'COL_NAMES': ['label', 'data'],
 'DESCR': 'mldata.org dataset: mnist-original',
 'data': array([[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]], dtype=uint8),
 'target': array([0., 0., 0., ..., 9., 9., 9.])}
```

ตัวแปร mnist จัดเก็บในรูปแบบของ dictionary โดยจะให้เก็บข้อมูลที่มีจำนวน 70,000 รูปภาพ โดยที่รูปภาพที่ 1-60,000 ใช้สำหรับการเรียนรู้ (Training Set) และรูปภาพที่ 60,001-70,000 ใช้สำหรับการทดสอบ (Test Set) แต่ละรูปภาพถูกกำหนดให้มีขนาด 28x28 พิกเซล และแต่ละพิกเซลจะเก็บค่า 0 (white) และ 255 (black)

รายละเอียดของ key ที่อยู่ในตัวแปร mnsit แสดงดังต่อไปนี้

data	ใช้เก็บข้อมูลพิกเซลของรูปภาพทั้งหมด
target	ใช้เก็บ target / class / label ของรูปภาพตัวเลข

จากนั้นสร้างตัวแปร X,y เพื่อจัดเก็บข้อมูลจากตัวแปร mnsit เพื่อนำไปใช้งาน

```
X,y = mnist['data'], mnist['target']
X.shape, y.shape
```

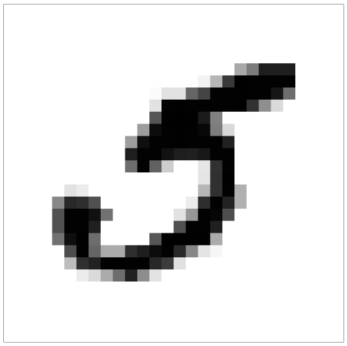
```
((70000, 784), (70000,))
```

โดยตัวแปร X ใช้สำหรับจัดเก็บข้อมูล (data) ตัวเลขทั้งหมด และตัวแปร y ใช้สำหรับเก็บข้อมูล label

จากตัวอย่างข้างต้น ตัวแปร x จะมีขนาด (70000, 784) นั่นหมายถึงมีจำนวน 70,000 ตัวเลข แต่ละตัวเลขมีขนาด 784 attribute (28x28) และตัวแปร y มีขนาด 70000 label ซึ่งต้องสัมพันธ์กับจำนวนของตัวแปร x

ตัวเลขแต่ละตัวถูกจัดเก็บเป็นแบบ vector ดังนั้น การแสดงข้อมูลจะต้องแปลงกลับให้เป็น array 2 มิติโดยใช้คำสั่ง reshape จากนั้นจึงจะสามารถแสดงข้อมูลแบบ Visualization ได้

```
import matplotlib.pyplot as plt
some_digit = X[36000]
some_digit_image = some_digit.reshape(28, 28)

plt.imshow(
    some_digit_image,
    cmap = plt.cm.binary,
    interpolation="nearest")

plt.axis("off")
plt.show()

y[36000]
```



```
5.0
```

เพื่อสะดวกต่อการแสดงผล สามารถสร้างเป็นฟังก์ชันในการ Visualization ข้อมูลดังตัวอย่างต่อไปนี้

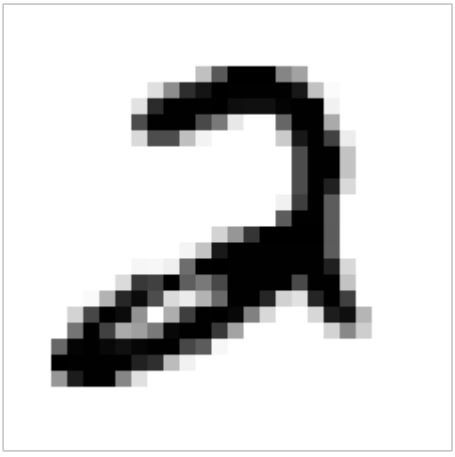
```
import matplotlib.pyplot as plt

def plot_digit(X, y):
    digit_image = X.reshape(28, 28)

    plt.imshow(
        digit_image,
        cmap = plt.cm.binary,
        interpolation="nearest")

    plt.axis("off")
    plt.show()

    print("Label", y)
```

โดย

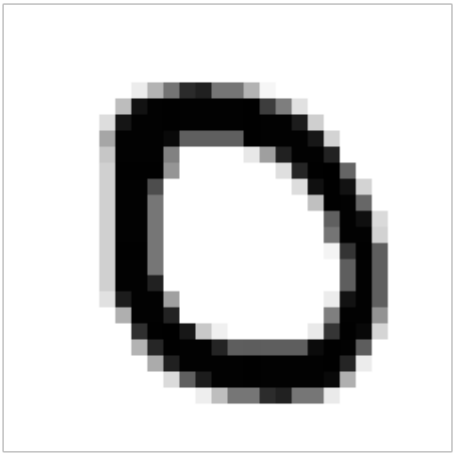
def	คือการประกาศฟังก์ชันในภาษา Python
plot_digit	คือชื่อของฟังก์ชัน
plot_digit(X, y)	คือการประกาศว่าฟังก์ชัน plot_digit กำหนดให้รับค่าพารามิเตอร์จำนวน 2 ค่า คือ X และ y

จากนั้นสามารถเรียกใช้ฟังก์ชัน plot_digit ดังตัวอย่างต่อไปนี้

```
plot_digit(X[15000], y[15000])
plot_digit(X[2000], y[2000])
```



```
('Label', 2.0)
```

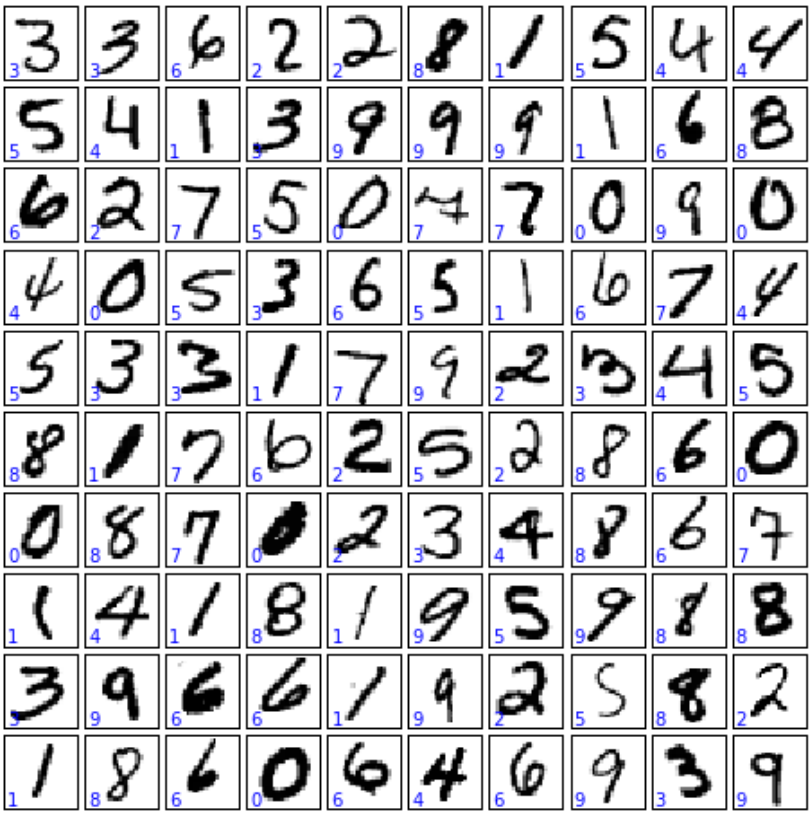


```
('Label', 0.0)
```

การ Visualization ชุดข้อมูล MNIST

ชุดข้อมูล MNIST เป็นชุดข้อมูลตัวเลขอารบิก ที่มีจำนวน 70,000 ตัวเลข หากต้องการที่จะ Visualization เพื่อให้เห็นข้อมูลได้อย่างชัดเจน และง่ายต่อการเข้าใจสามารถทำได้โดย คำสั่งต่อไปนี้

```
import matplotlib.pyplot as plt
fig, axes = plt.subplots(10, 10, figsize=(8, 8),
                  subplot_kw={'xticks':[], 'yticks':[]},
                  gridspec_kw=dict(hspace=0.1, wspace=0.1))

for i, ax in enumerate(axes.flat):
    ax.imshow(X_train[i].reshape((28,28)), cmap='binary',
interpolation='nearest')
    ax.text(0.05, 0.05, str(int(y_train[i])),
           transform=ax.transAxes, color='blue')
```



ภาพประกอบที่ 8: การ Visualization ชุดข้อมูล MNIST พร้อมทั้งแสดง label ของแต่ละตัวเลข

จากภาพประกอบที่ 8 แสดงให้เห็นถึงการ Visualization ข้อมูลตัวเลขทั้งสิ้น 100 รูปภาพ พร้อมทั้งแสดง target/class ที่ถูกต้องของข้อมูลทั้งหมด

บทที่ 3
ข้อมูลชุดเรียนรู้และข้อมูลชุดทดสอบ (Training and Test Data)

กระบวนการเรียนรู้ของเครื่องจักรจะต้องแบ่งข้อมูลออกเป็นสองส่วน ประกอบด้วย ข้อมูลชุดเรียนรู้ (Training Set) และข้อมูลชุดทดสอบ (Test Set) ซึ่งข้อมูลชุดเรียนรู้จะถูกนำไปเรียนด้วยด้วยวิธีการเรียนรู้เครื่องจักรเพื่อสร้างออกมาเป็นโมเดล (Model) จากนั้นจึงทดสอบโมเดลที่สร้างด้วยข้อมูลชุดทดสอบ หากโมเดลที่สร้างมีประสิทธิภาพดีจึงนำโมเดลนั้นไปใช้งานจริง บางชุดข้อมูลอาจไม่ได้แบ่งข้อมูลออกเป็นสองชุดที่ชัดเจน ดังนั้นจึงมีความจำเป็นที่จะต้องเขียนโปรแกรมเพื่อแบ่งข้อมูลออกเป็นสองส่วน

โปรแกรม scikit-learn มีเครื่องมือที่ช่วยในการแบ่ง (Split) และสลับ (Shuffle) ข้อมูลโดยใช้ฟังก์ชันดังต่อไปนี้

```
from sklearn.model_selection import train_test_split
# Loading iris dataset
from sklearn.datasets import load_iris
iris_dataset = load_iris()

# split and shuffle data
X_train, X_test, y_train, y_test =  \
     train_test_split(iris_dataset['data'], \
     iris_dataset['target'], random_state=0)
```

จากตัวอย่างข้างต้น เมื่อใช้คำสั่ง train_test_split โปรแกรมจะแบ่งข้อมูล และนำมาจัดเก็บลงไปในตัวแปลจำนวน 4 ตัวประกอบด้วย X_train, X_test, y_train และ y_test โดยเป็นข้อมูลประเภท numpy.ndarray โปรแกรมจะแบ่งข้อมูลออกเป็น 75% สำหรับเรียนรู้ และเก็บอยู่ในตัวแปร X_train และ y_train ส่วนที่เหลืออีก 25% สำหรับทดสอบ จะถูกจัดเก็บในตัวแปร X_test และ y_test

สามารถเรียกดูขนาดของข้อมูลด้วยคำสั่ง shape ดังตัวอย่างต่อไปนี้

```
print("X_train shape: {}".format(X_train.shape))
print("y_train shape: {}".format(y_train.shape))
print("X_test shape: {}".format(X_test.shape))
print("y_test shape: {}".format(y_test.shape))
```

```
X_train shape: (112, 4)
y_train shape: (112,)
X_test shape: (38, 4)
y_test shape: (38,)
```

การกำหนดอัตราที่ใช้สำหรับแบ่งข้อมูล

โปรแกรม scikit-learn อนุญาตให้ผู้ใช้แบ่งขนาดของข้อมูลชุดเรียนรู้และชุดทดสอบได้ โดยใช้ keyword คือ test_size สามารถทำได้ดังต่อไปนี้

```
# กำหนดให้ test set มีจำนวนข้อมูล 50% จากข้อมูลทั้งหมด
X_train, X_test, y_train, y_test =
     train_test_split(iris_dataset['data'], \
     iris_dataset['target'], test_size=0.5, \
random_state=0)
```

จากตัวอย่างกำหนดให้ test_size = 0.5 คือการระบุขนาดของ Train set และ Test set ให้มีขนาดเป็น 50% สามารถตรวจสอบขนาดของข้อมูลได้ดังนี้

```
print("X_train shape: {}".format(X_train.shape))
print("y_train shape: {}\n".format(y_train.shape))
print("X_test shape: {}".format(X_test.shape))
print("y_test shape: {}".format(y_test.shape))
```

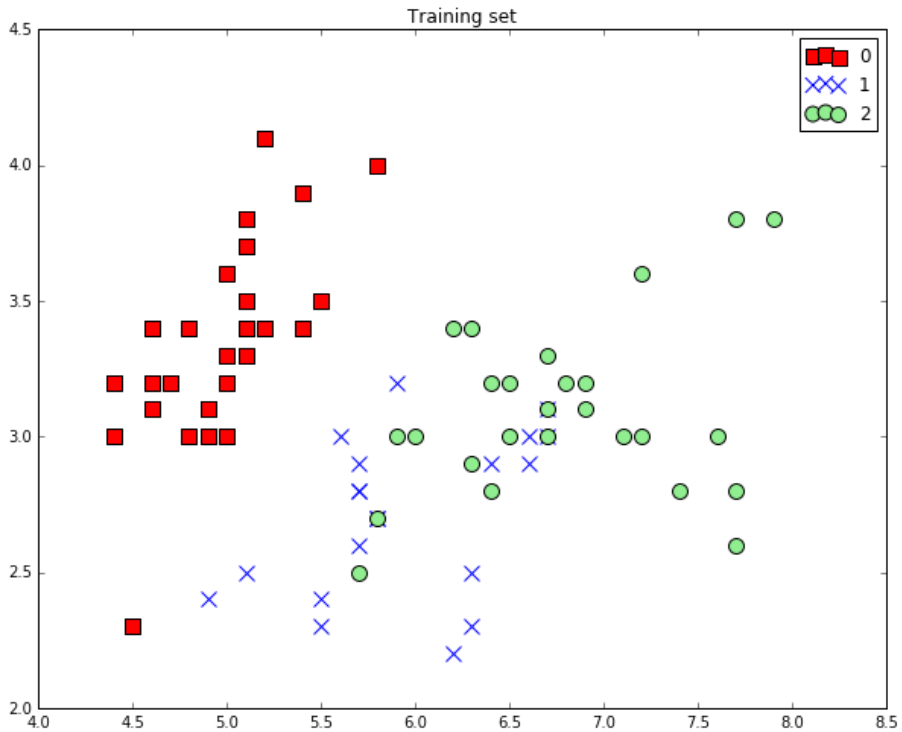
```
X_train shape: (75, 4)
y_train shape: (75,)
X_test shape: (75, 4)
y_test shape: (75,)
```

การแสดงข้อมูลในรูปแบบของกราฟ

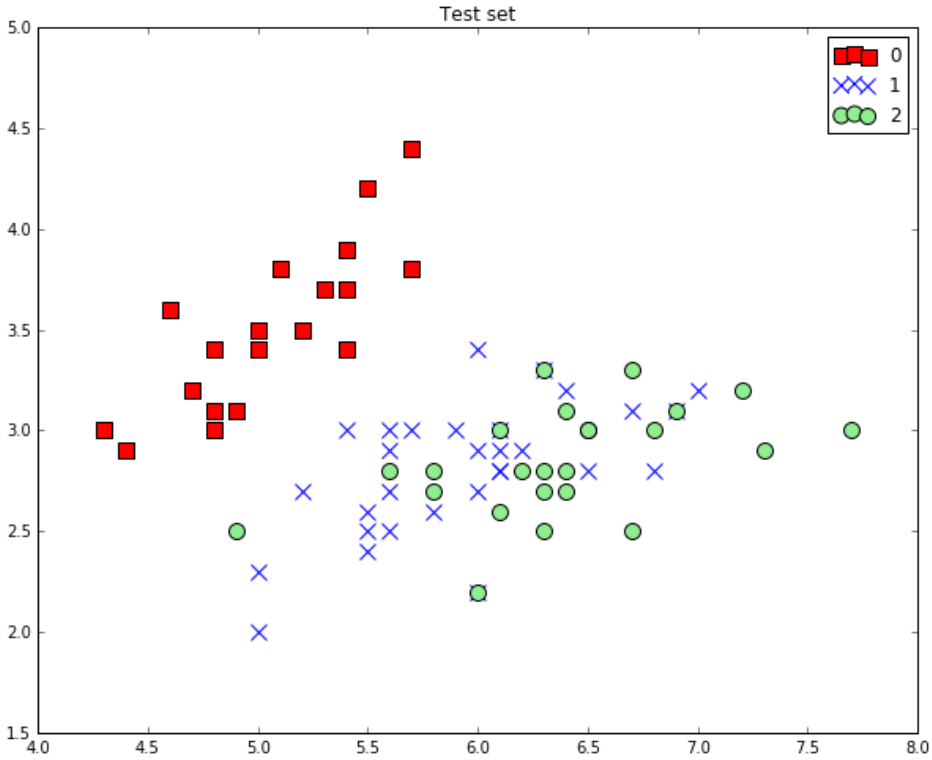
จากตัวอย่างข้างต้นได้กำหนด test_size = 50 ทำให้แบ่งข้อมูลออกเป็น Training set และ Test set ชุดละเท่า ๆ กัน ข้อมูลทั้งสองส่วนสามารถ Visualization ให้เห็นข้อมูลเข้าใจขึ้น โดยแสดงในลักษณะของกราฟ โดยกราฟที่จะนำเสนอเป็นกราฟ 2 มิติ ประกอบด้วยแกน X และแกน Y ดังนั้น จึงต้องเลือกข้อมูลจากชุดข้อมูล iris ที่มีทั้งหมด 4 attribute มาแสดงเพียง 2 attribute

การแสดงข้อมูล Training set สามารถทำได้ดังนี้

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

markers = ('s', 'x', 'o')
colors = ('red', 'blue', 'lightgreen')
cmap = ListedColormap(colors[:len(np.unique(y_test))])
plt.figure(figsize=(10, 8))

for idx, cl in enumerate(np.unique(y_train)):
    plt.scatter(x=X_train[y_train == cl, 0],
                y=X_train[y_train == cl, 1],
                c=cmap(idx), marker=markers[idx],
                label=cl, s=100)

plt.title('Training set')
plt.legend()
plt.show()
```



เช่นเดียวกับข้อมูล Test set สามารถแสดงในรูปแบบของกราฟ โดยต้องเปลี่ยนตัวแปรจาก X_train และ y_train ให้เป็น X_test และ y_test ตามลำดับ การแสดงข้อมูล Test set แสดงดังตัวอย่างต่อไปนี้



บทที่ 4
ไลบรารี่ Matplotlib (Matplotlib Library)

Matplotlib เป็นไลบรารี่ที่ใช้ในโปรแกรมภาษา Python สำหรับการ Visualize ข้อมูล (Data Visualization) หรือพลอต (Plot) ข้อมูลทั้งในรูปแบบ 2 มิติ และ 3 มิติ การติดตั้งไลบรารี่ Matplotlib ในลินุกซ์ (Linux) สามารถทำได้โดยพิมพ์คำสั่งลงไปในเทอร์มินัล ดังนี้

```
$ pip install matplotlib
```

ทั้งนี้ยังสามารถติดตั้งผ่าน Jupyter โดยพิมพ์คำสั่งดังต่อไปนี้

```
$ ! pip install matplotlib
```

หลังจากติดตั้ง Matplotlib เสร็จเรียบร้อยสามารถเรียกใช้ภายในโปรแกรมภาษา Python โดยใช้คำสั่งดังต่อไปนี้เพื่ออิมพอร์ตไลบรารี่ Matplotlib เข้ามาใช้งาน

```
import matplotlib.pyplot as plt
```

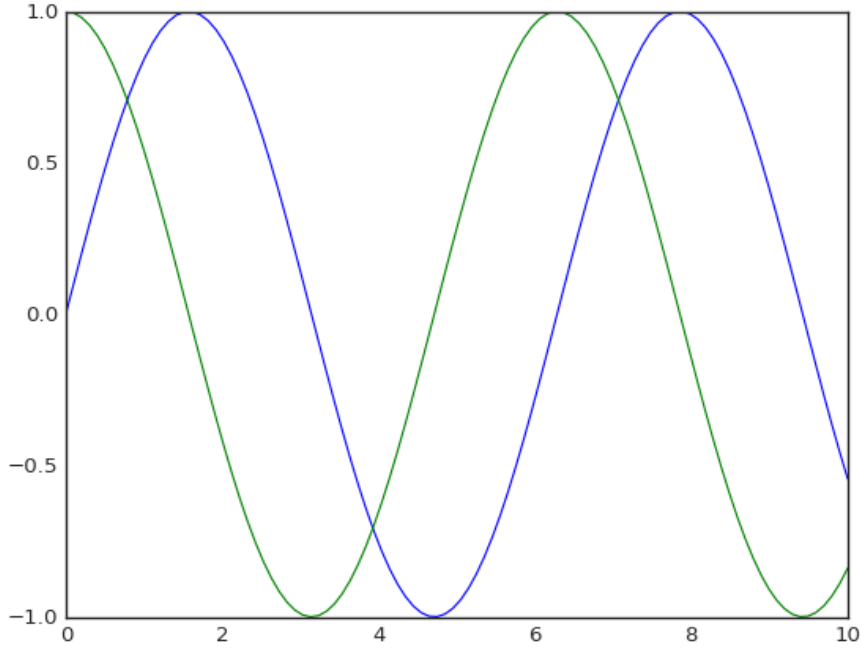
ตัวอย่างของการสร้างกราฟโดยใช้ Matplotlib แสดงดังตัวอย่างต่อไปนี้

```
# ----------file: my_plot.py ------------
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))

plt.show()
```

จากตัวอย่างข้างต้นเป็นสคริปต์ (Script) ภาษา Python ที่บันทึกไว้ในไฟล์ชื่อ my_plot.py ดังนั้น หากต้องการจะเรียกใช้ไฟล์ภาษา Python เพื่อทำงาน (Run) จะต้องพิมพ์คำสั่งที่ Terminal ดังตัวอย่างต่อไปนี้

```
$ python my_plot.py
```

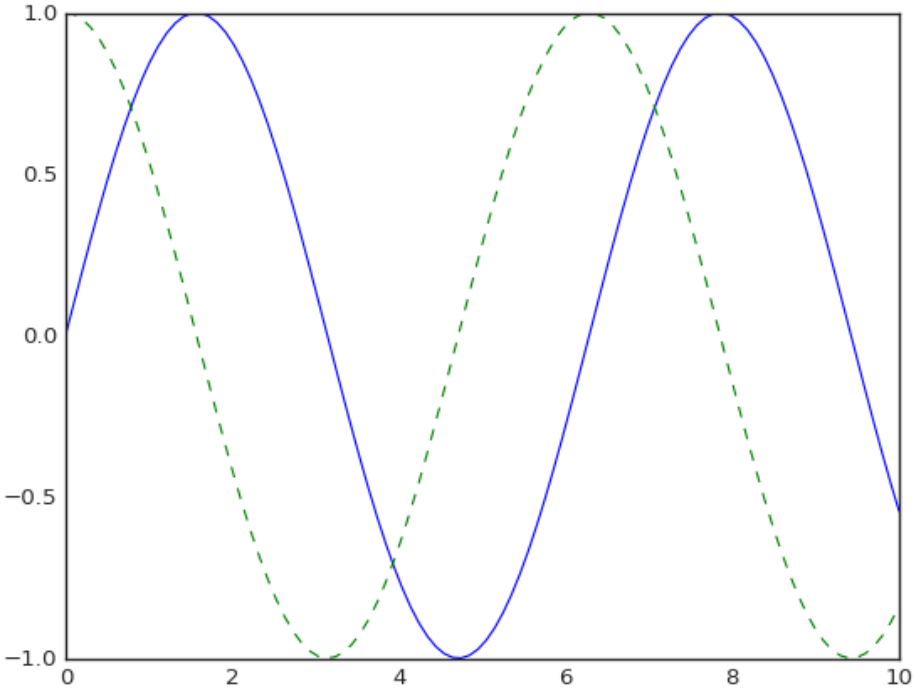
โปรแกรมจะแสดงกราฟออกทางจอภาพ ดังตัวอย่างต่อไปนี้



โปรแกรม Matplotlib อนุญาตให้ทำการปรับเปลี่ยนรูปแบบของกราฟได้ตามต้องการ ดังตัวอย่างต่อไปนี้

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 100)

fig = plt.figure()
plt.plot(x, np.sin(x), '-')
plt.plot(x, np.cos(x), '--')

plt.show()
```

ผลลัพธ์ที่ได้แสดงดังตัวอย่างต่อไปนี้



การบันทึกกราฟ (Figure)

ในโปรแกรม Matplotlib สามารถบันทึกกราฟให้เป็นรูปภาพ โดยใช้คำสั่ง savefig() โดยการบันทึก จะบันทึกในรูปแบบของไฟล์ PNG ตัวอย่างการบันทึกแสดงดังตัวอย่างต่อไปนี้

```
fig.savefig('my_figure.png')
```

จากตัวอย่างข้างต้น เป็นการบันทึกรูปภาพเก็บลงไปในโฟลเดอร์ หากต้องการที่จะตรวจสอบรูปภาพ (ในกรณีนี้ชื่อที่ถูกกำหนดคือ my_figure.png) จากนั้นสามารถตรวจสอบรูปภาพ my_figure.png ที่บันทึกลงไป สามารรถทำได้โดยใช้คำสั่ง ls เพื่อดูข้อมูล แต่ในกรณีที่ต้องการดูผ่าน Jupyter Notebook จะต้องใช้เครื่องหมาย "!” เข้ามาช่วย ดังนั้น เครื่องหมาย "!” จึงเปรียบเสมือนการทำงานผ่านหน้าจอเทอร์มินัล

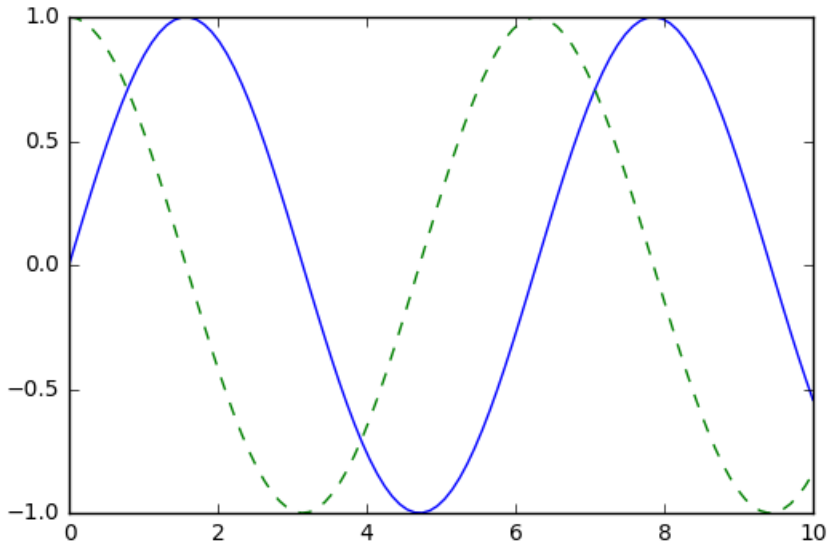
```
! ls -lh my_figure.png
```

```
-rw-rw-r-- 1 mrolarik mrolarik 26K Jul 19 00:12
my_figure.png
```

การแสดงรูปภาพ (Image Show)

ในโปรแกรม Python สามารถเลือกใช้ไลบรารี่ต่าง ๆ เพื่อใช้สำหรับการเปิดรูปภาพจากไฟล์แสดงดังต่อไปนี้

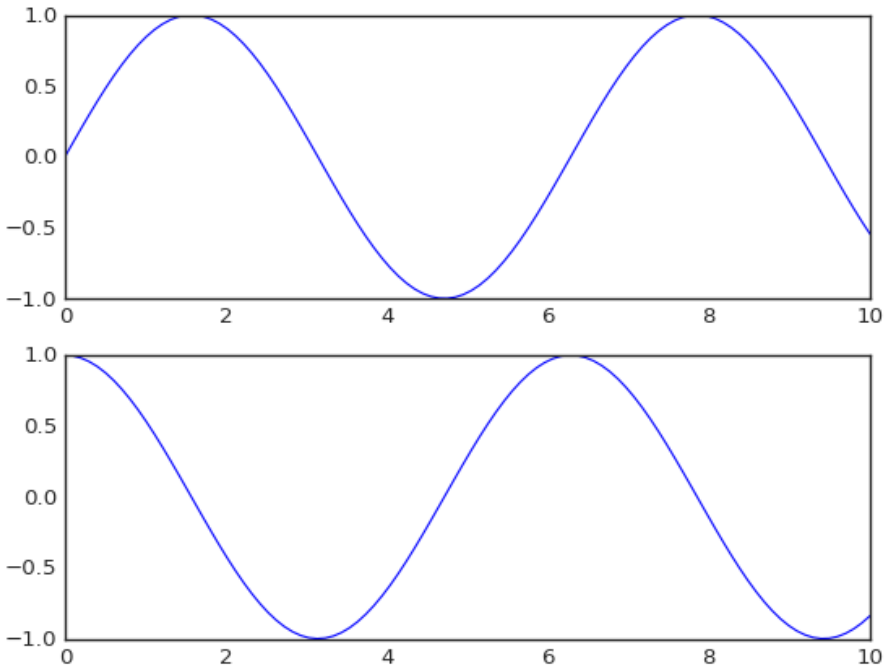
```
from IPython.display import Image
Image('my_figure.png')
```



วิธีการแสดงรูปภาพโดยใช้อินเทอร์เฟซ (Interface) ที่ต่างกัน

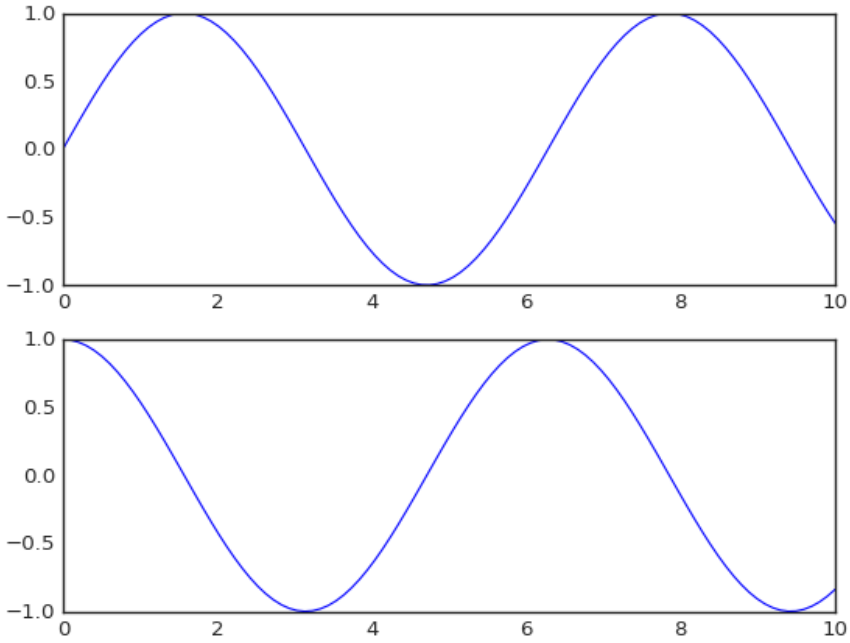
Matlab-style Interface

```
plt.figure()    # สร้าง plot figure

# สร้าง panel แรกจากทั้งหมด 2 panel และเซ็ต axis ให้กับกราฟแรก
plt.subplot(2, 1, 1) # (rows, columns, panel number)
plt.plot(x, np.sin(x))

# สร้าง panel ที่ 2 และเซ็ต axis
plt.subplot(2, 1, 2)
plt.plot(x, np.cos(x))

plt.show()
```



Object-oriented Interface

```
# สร้าง grid
# ax จะเป็น array ของ Axes object
fig, ax = plt.subplots(2) # สร้าง Axes จำนวน 2 object

# เรียกใช้เมธอด (Method) plot()
ax[0].plot(x, np.sin(x))
ax[1].plot(x, np.cos(x))

plt.show()
```

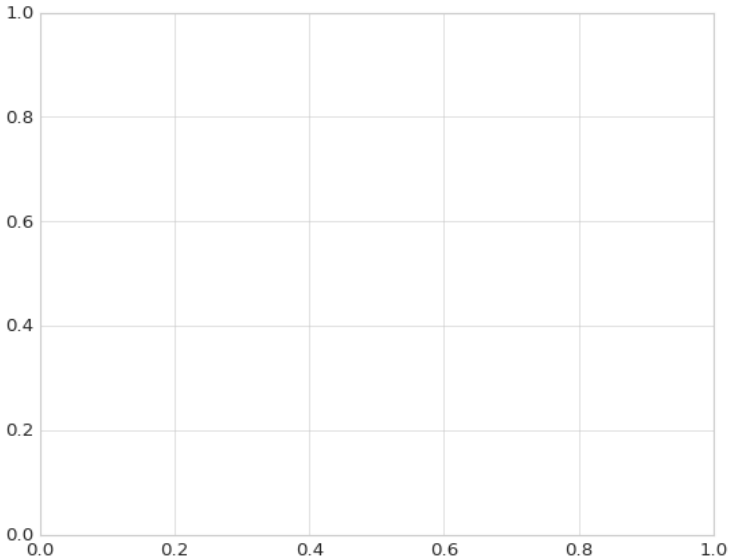


การเพิ่มเส้นตาราง (Grid) ในการพลอต

การเพิ่มเส้นตาราง หรือ Grid ในการพลอตทำได้โดยใช้คำสั่ง plt.style.use('seaborn-whitegrid') แสดงดังต่อไปนี้

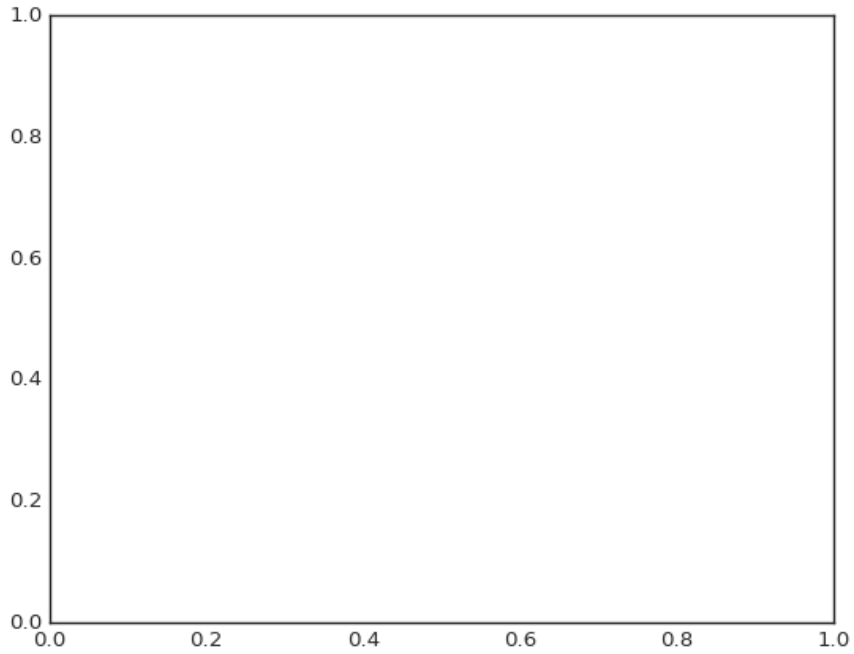
```
import numpy as np
import matplotlib.pyplot as plt

plt.style.use('seaborn-whitegrid')
fig = plt.figure()
ax = plt.axes()
```

ผลลัพธ์ที่ได้คือกราฟที่แสดงจะมีเส้น Grid ทั้งในแนวตั้งและแนวนอน



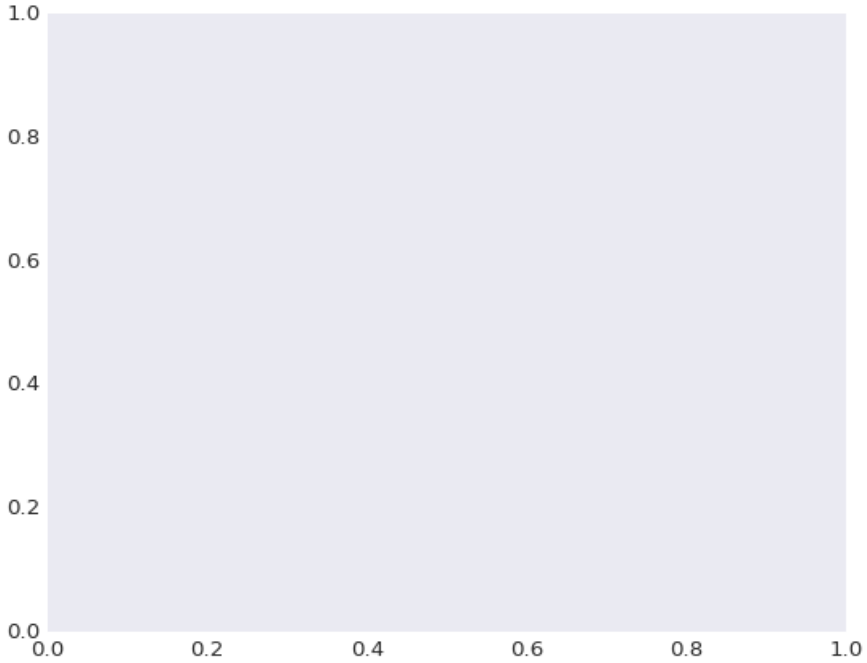
หากไม่ต้องการแสดงเส้น Grid สามารถใช้คำสั่ง plt.style.use('seaborn-white') ดังตัวอย่างต่อไปนี้

```
plt.style.use('seaborn-white')
fig = plt.figure()
ax = plt.axes()

plt.show()
```



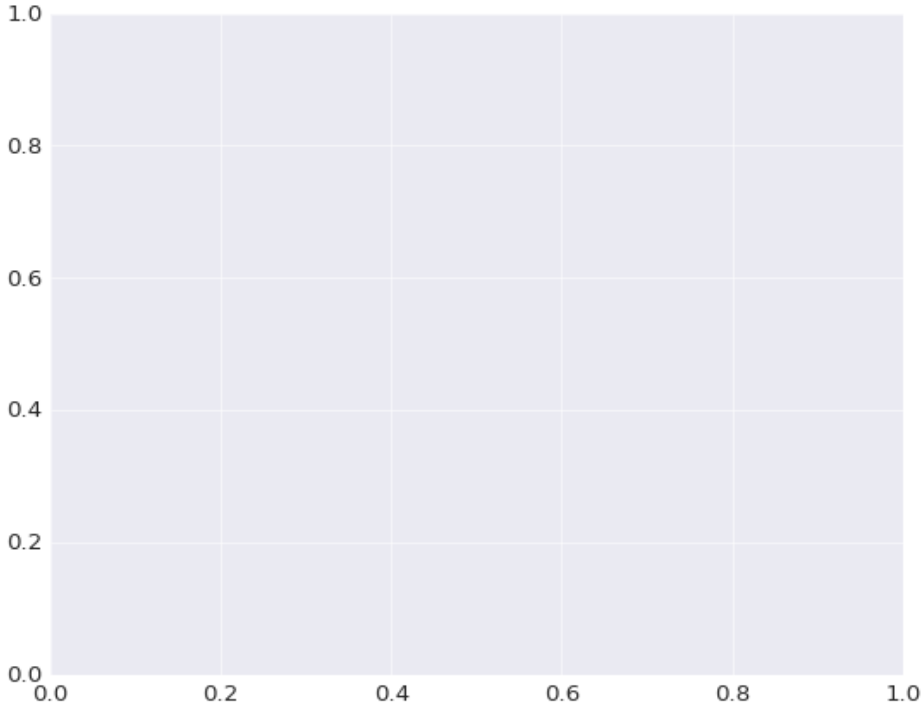
หากต้องการให้พื้นหลังมืด (Dark) สามารถใช้คำสั่ง plt.style.use('seaborn-dark') พื้นหลังของกราฟจะเปลี่ยนจากสีขาวเป็นโทนสีฟ้า

```
plt.style.use('seaborn-dark')
fig = plt.figure()
ax = plt.axes()

plt.show()
```



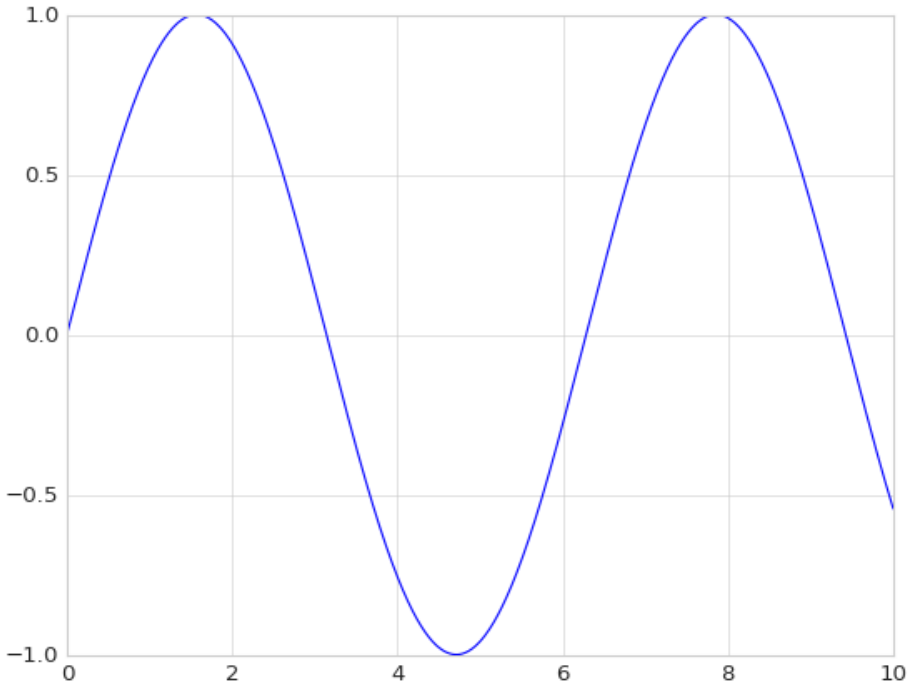
หากต้องการให้พื้นหลังมืด (Dark) และมีเส้น Grid สามารถใช้คำสั่ง plt.style.use('seaborn-darkgrid')

```
plt.style.use('seaborn-darkgrid')
fig = plt.figure()
ax = plt.axes()

plt.show()
```



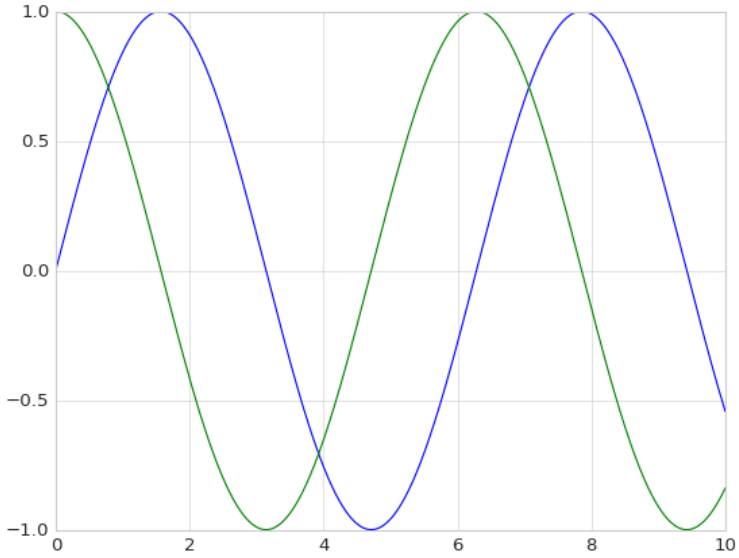
ตัวอย่างการพลอตกราฟเส้น

```
plt.style.use('seaborn-whitegrid')
fig = plt.figure()
ax = plt.axes()

x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x))
plt.show()
```



หากต้องการให้พลอตจำนวนสองเส้นใน Figure เดียวกัน สามารถทำได้โดยใช้คำสั่ง plt.plot() ได้มากกว่า 1 ครั้ง ดังต่อไปนี้

```
plt.style.use('seaborn-whitegrid')
plt.figure()
plt.axes()

x = np.linspace(0, 10, 1000)
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))

plt.show()
```



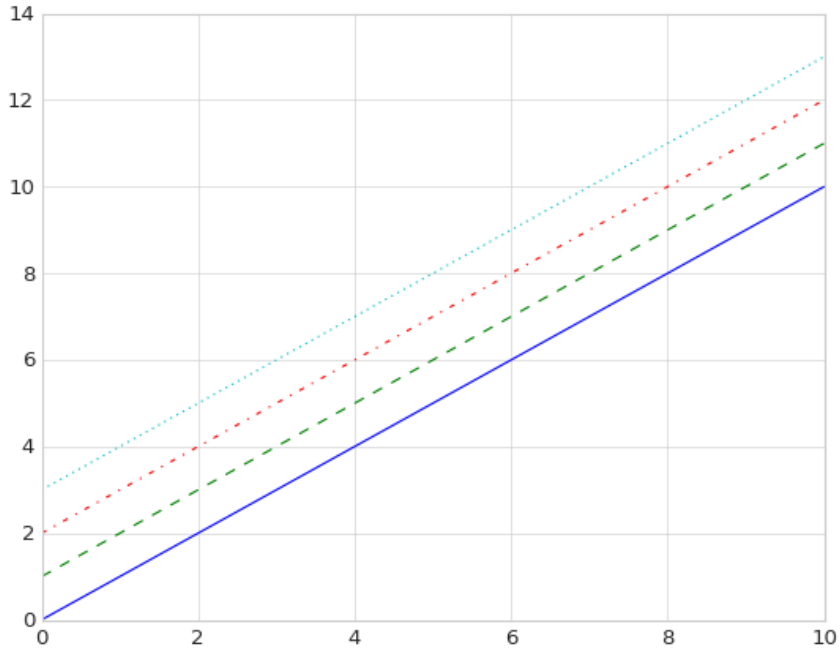
การกำหนดวิธีการพลอต: Line Color และ Style

กำหนด Line Color

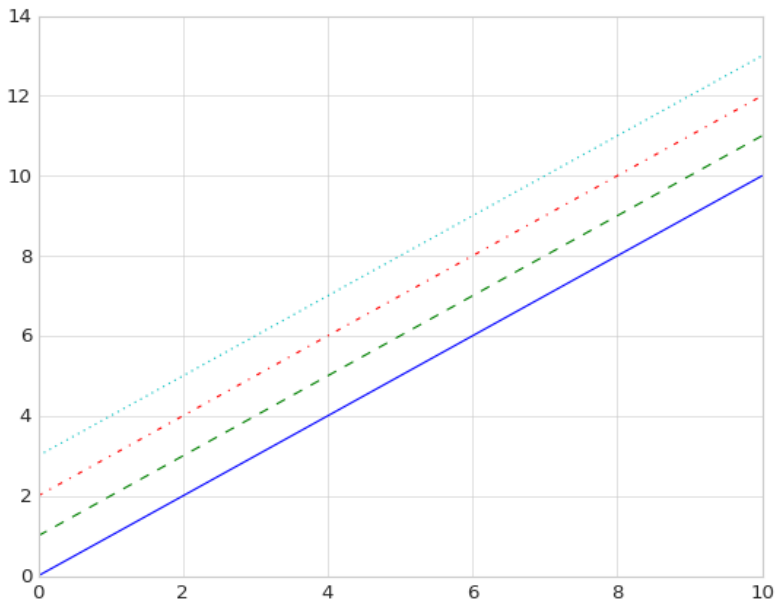
```
plt.style.use('seaborn-whitegrid')
plt.plot(x, np.sin(x-0), color='blue')
                                # กำหนดสีโดยใช้ชื่อ
plt.plot(x, np.sin(x-1), color='g')
                                # กำหนดสีโดยใช้ชื่อย่อ จากโหมดสี rgbcmyk
plt.plot(x, np.sin(x-2), color='0.75')
                                # กำหนดสีโดยใช้ค่าสีเทา ระหว่าง 0 ถึง 1
plt.plot(x, np.sin(x-3), color='#FFDD44')
                                # กำหนดสีโดยใช้ Hex code (RRGGBB จาก 00 ถึง FF)
plt.plot(x, np.sin(x-4), color=(1.0,0.2,0.3))
                                # กำหนดสีโดยใช้ค่าสี RGB tuple, ค่าจาก 0 ถึง 1
plt.plot(x, np.sin(x-5), color='chartreuse')
                                # กำหนดสีโดยใช้ค่าสี HTML
plt.show()
```

กำหนด Line Style

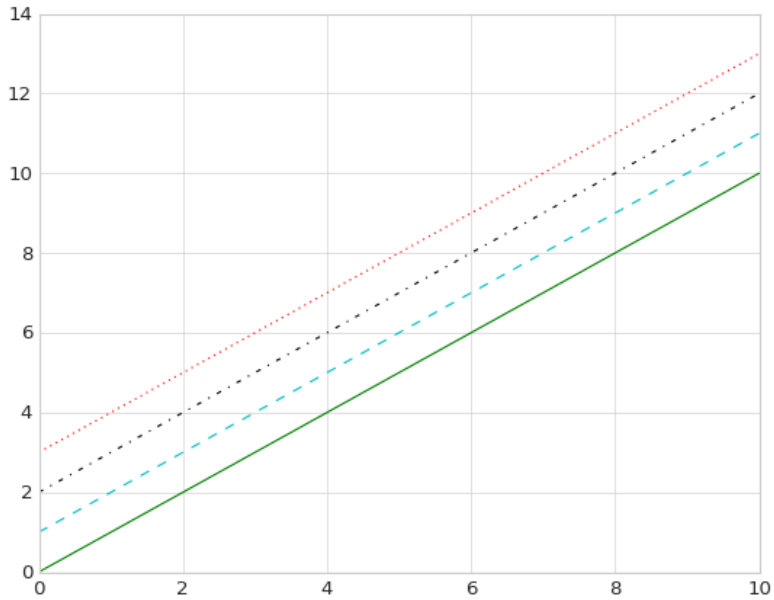
```
plt.plot(x, x + 0, linestyle='solid')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
plt.plot(x, x + 3, linestyle='dotted')
plt.show()
```



```
plt.plot(x, x + 0, linestyle='-')       # solid
plt.plot(x, x + 1, linestyle='--')      # dashed
plt.plot(x, x + 2, linestyle='-.')      # dashdot
plt.plot(x, x + 3, linestyle=':')       # dotted
plt.show()
```

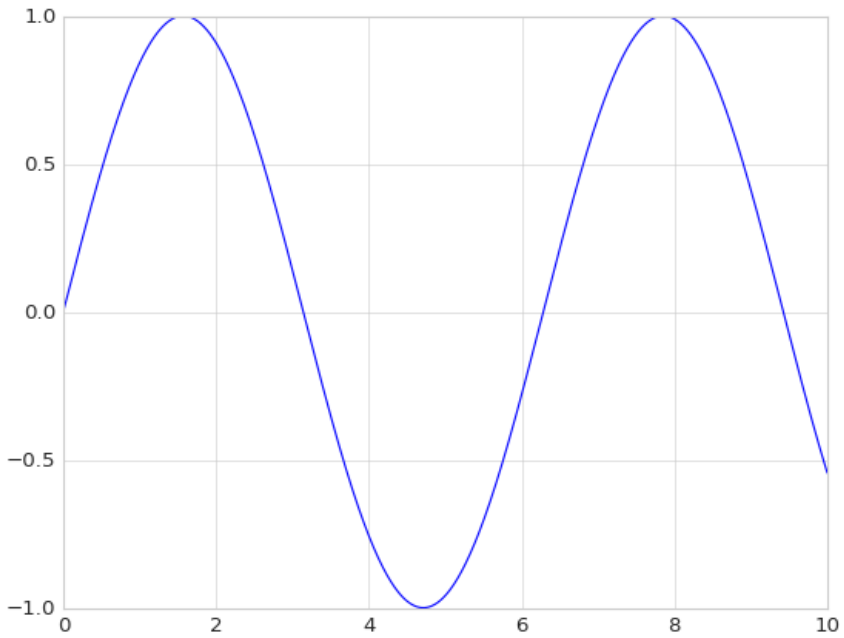



```
plt.plot(x, x + 0, '-g')      # solid green
plt.plot(x, x + 1, '--c')     # dashed cyan
plt.plot(x, x + 2, '-.k')     # dashdot black
plt.plot(x, x + 3, ':r')      # dotted red
plt.show()
```



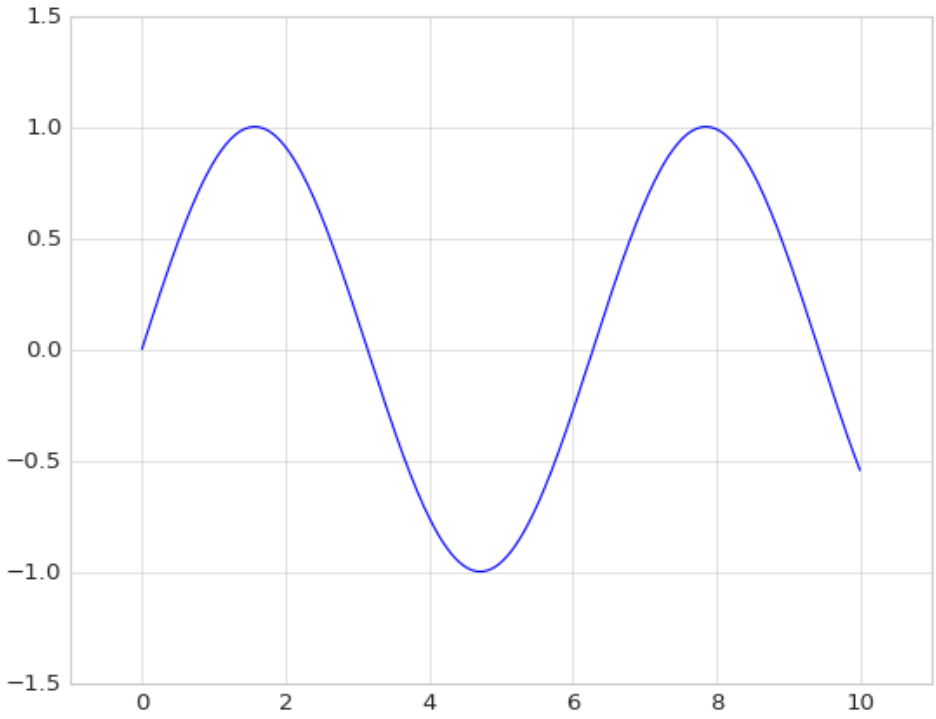
การกำหนด Axes Limit

```
plt.plot(x, np.sin(x))
plt.show()
```



จากตัวอย่างข้างต้น กราฟที่แสดงจะมีขนาดพอดีกับเส้นกราฟ หากต้องการกำหนด Axes Limit สามารถทำได้โดยใช้คำสั่ง plt.xlim(), plt.ylim() แสดงดังต่อไปนี้

```
plt.plot(x, np.sin(x))
plt.xlim(-1,11)
plt.ylim(-1.5, 1.5)
plt.show()
```

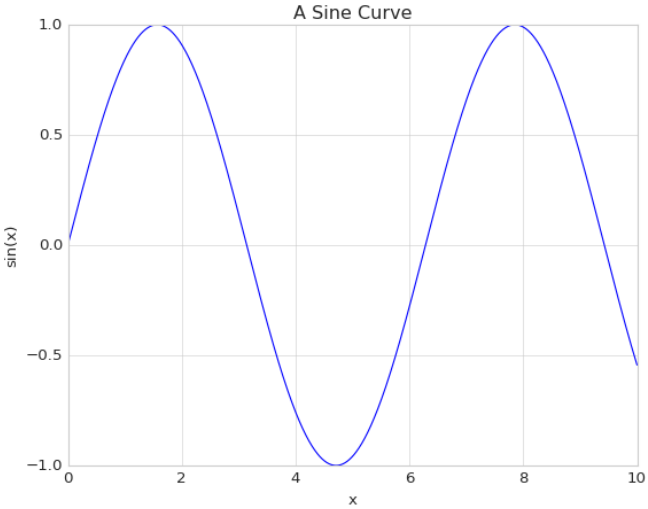


จากตัวอย่างข้างต้น Scale ของแกน X และแกน Y จะเปลี่ยนไปตามที่กำหนดไว้ใน plt.xlim(-1, 11) และ plt.ylim(-1.5, 1.5)

การพลอต Label

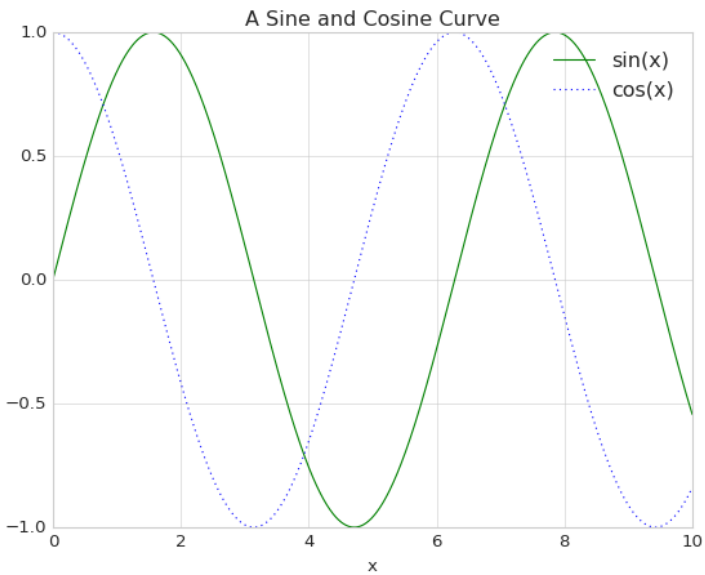
หากต้องการเพิ่ม Label ให้กับกราฟ เช่น title, x-axis, y-axis สามารถทำได้โดย

```
plt.plot(x, np.sin(x))
plt.title("A Sine Curve")
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.show()
```



การพลอต Legend

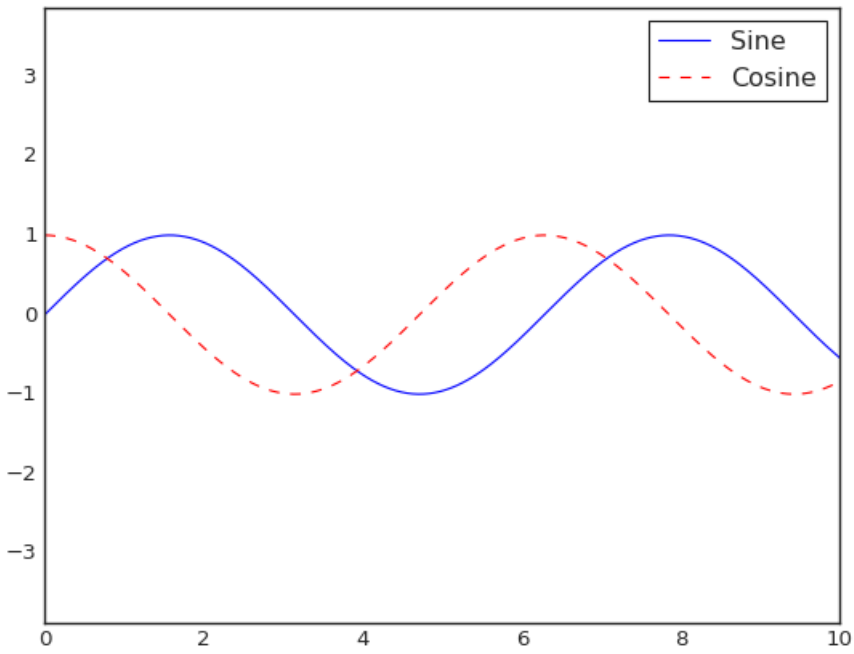
```
plt.plot(x, np.sin(x), '-g', label='sin(x)')
plt.plot(x, np.cos(x), ':b', label='cos(x)')
plt.title("A Sine and Cosine Curve")
plt.xlabel("x")
plt.legend()
plt.show()
```



การกำหนดการพลอต Legend

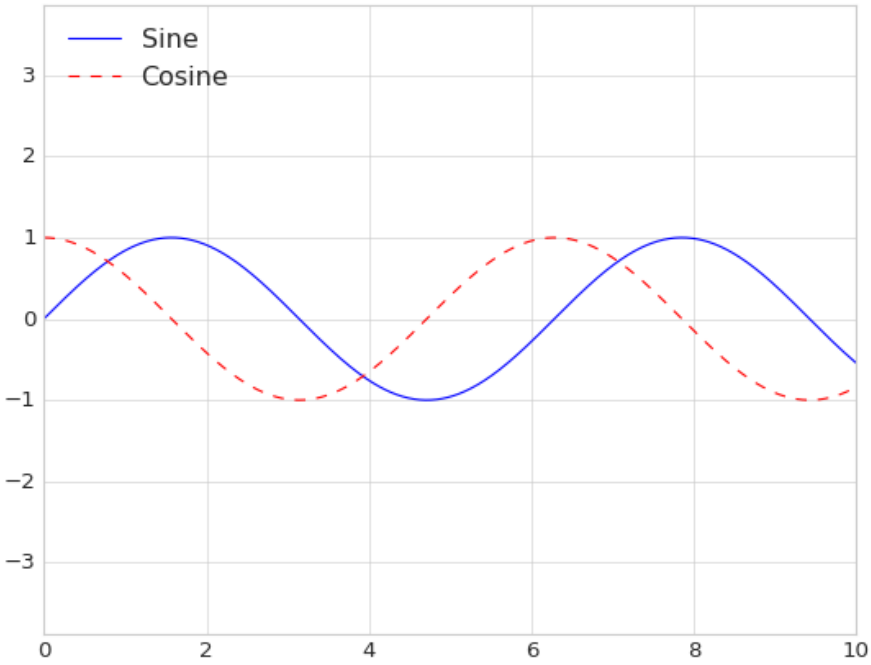
การเพิ่มกรอบในกับ Legeng

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')

x = np.linspace(0, 10, 1000)
fig, ax = plt.subplots()
ax.plot(x, np.sin(x), '-b', label='Sine')
ax.plot(x, np.cos(x), '--r', label='Cosine')
ax.axis('equal')
leg = ax.legend(frameon=True)
```



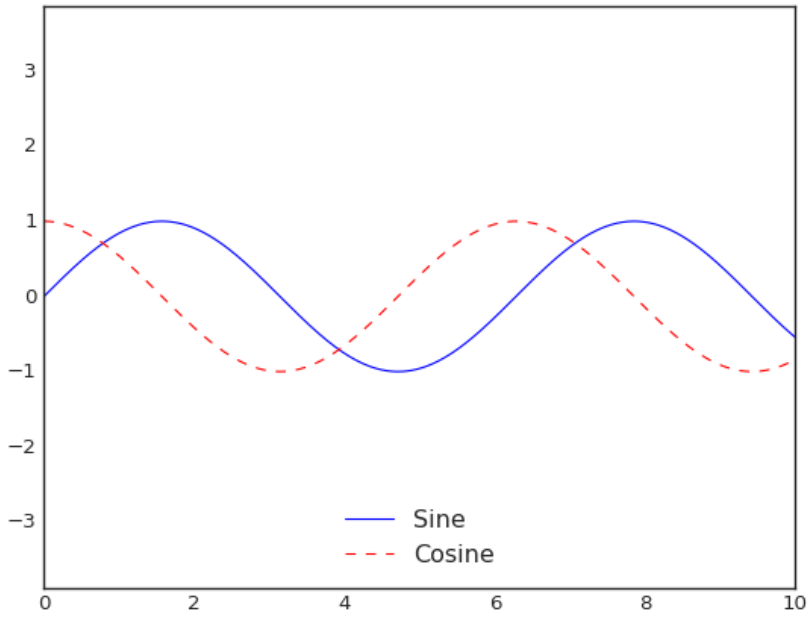
การเปลี่ยนตำแหน่ง Legend ให้อยู่มุมบนซ้าย

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')

x = np.linspace(0, 10, 1000)
fig, ax = plt.subplots()
ax.plot(x, np.sin(x), '-b', label='Sine')
ax.plot(x, np.cos(x), '--r', label='Cosine')
ax.axis('equal')
ax.legend(loc='upper left', frameon=False)
plt.show()
```

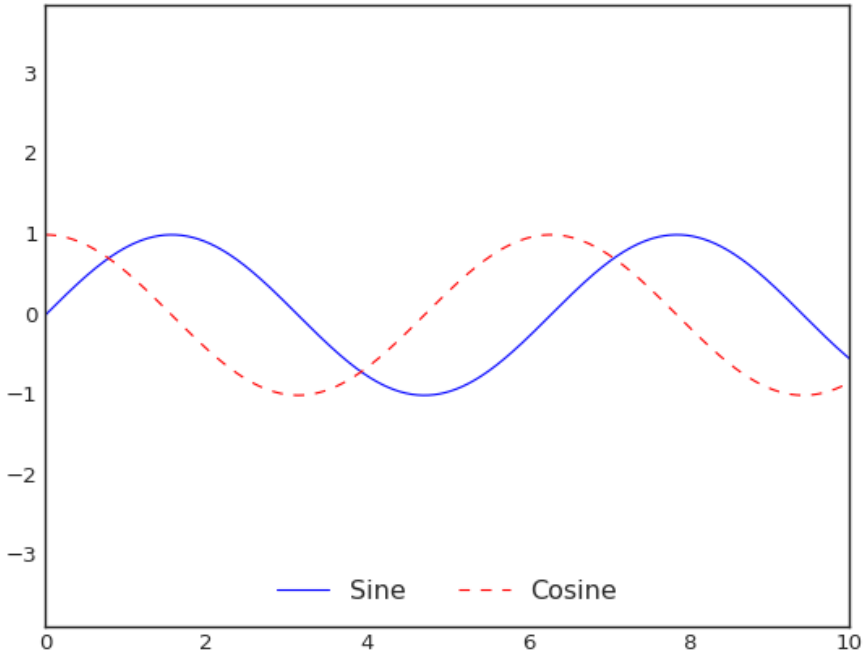


การเปลี่ยนตำแหน่ง Legend ให้อยู่ด้านล่างตรงกลาง

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')

x = np.linspace(0, 10, 1000)
fig, ax = plt.subplots()
ax.plot(x, np.sin(x), '-b', label='Sine')
ax.plot(x, np.cos(x), '--r', label='Cosine')
ax.axis('equal')
ax.legend(loc='lower center', frameon=False)
plt.show()
```

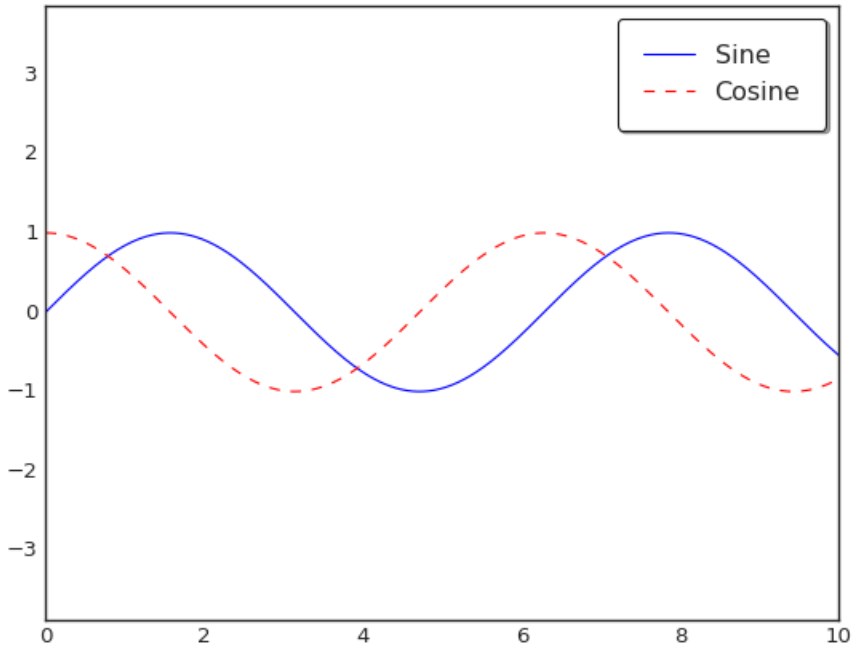


```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')

x = np.linspace(0, 10, 1000)
fig, ax = plt.subplots()
ax.plot(x, np.sin(x), '-b', label='Sine')
ax.plot(x, np.cos(x), '--r', label='Cosine')
ax.axis('equal')
ax.legend(loc='lower center', ncol=2, frameon=False)
plt.show()
```



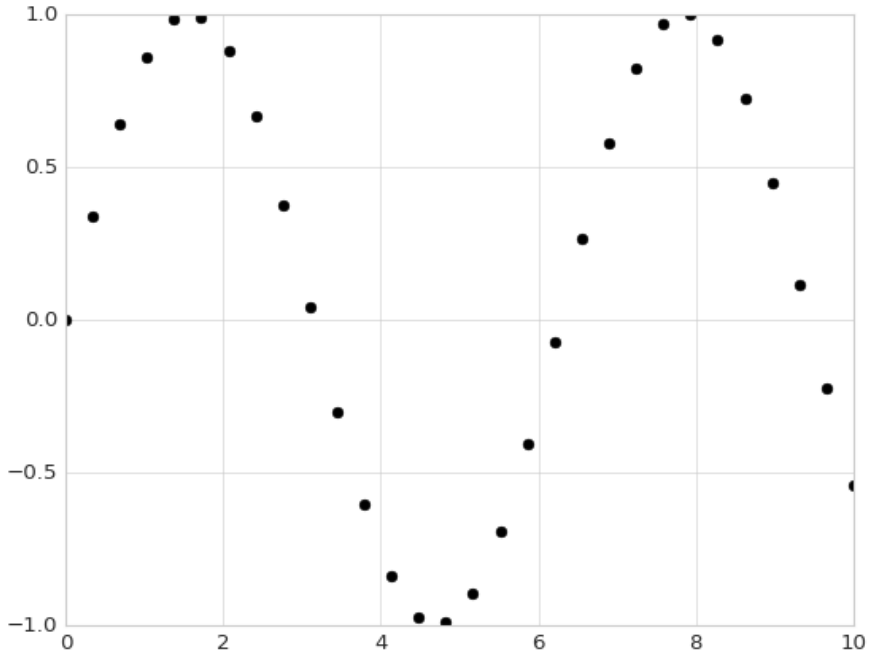
การตกแต่ง Legend

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')

x = np.linspace(0, 10, 1000)
fig, ax = plt.subplots()
ax.plot(x, np.sin(x), '-b', label='Sine')
ax.plot(x, np.cos(x), '--r', label='Cosine')
ax.axis('equal')
leg = ax.legend(frameon=True, fancybox=True, framealpha=1,
                    shadow=True, borderpad=1)
plt.show()
```

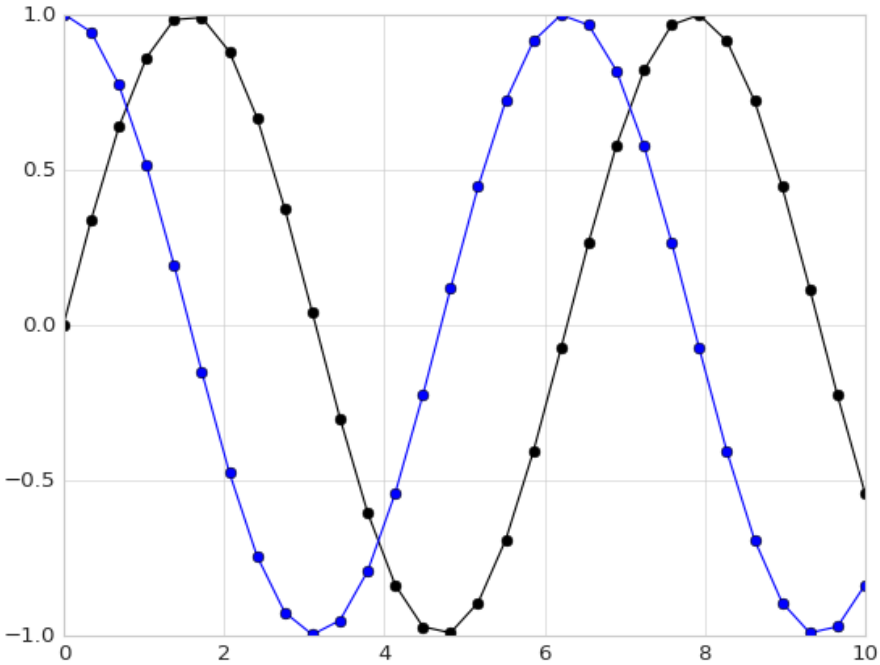


การพลอตแบบ Scatter

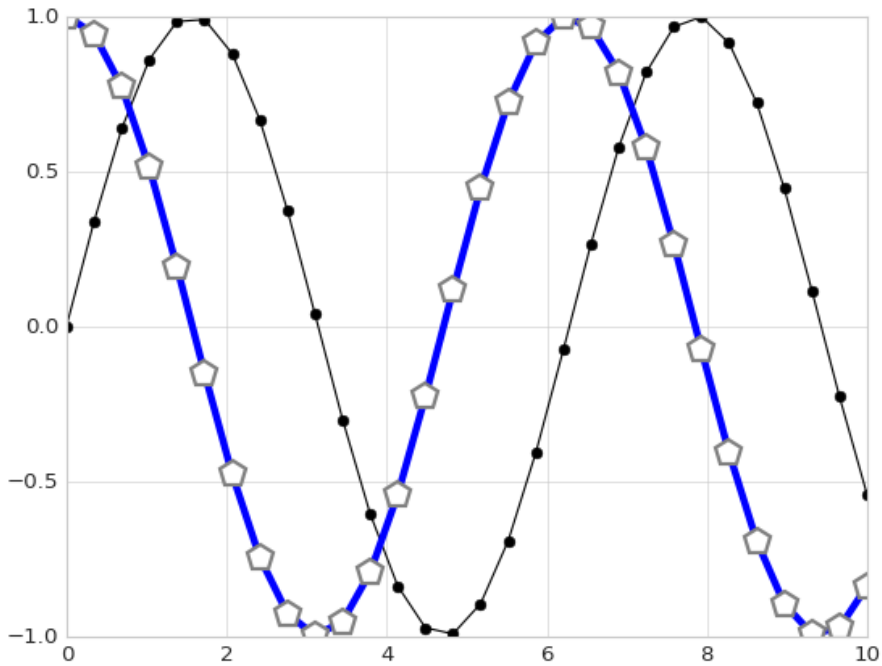
```
import numpy as np

x = np.linspace(0, 10, 30)
y = np.sin(x)

plt.plot(x, y, 'o', color='black')
plt.show()
```



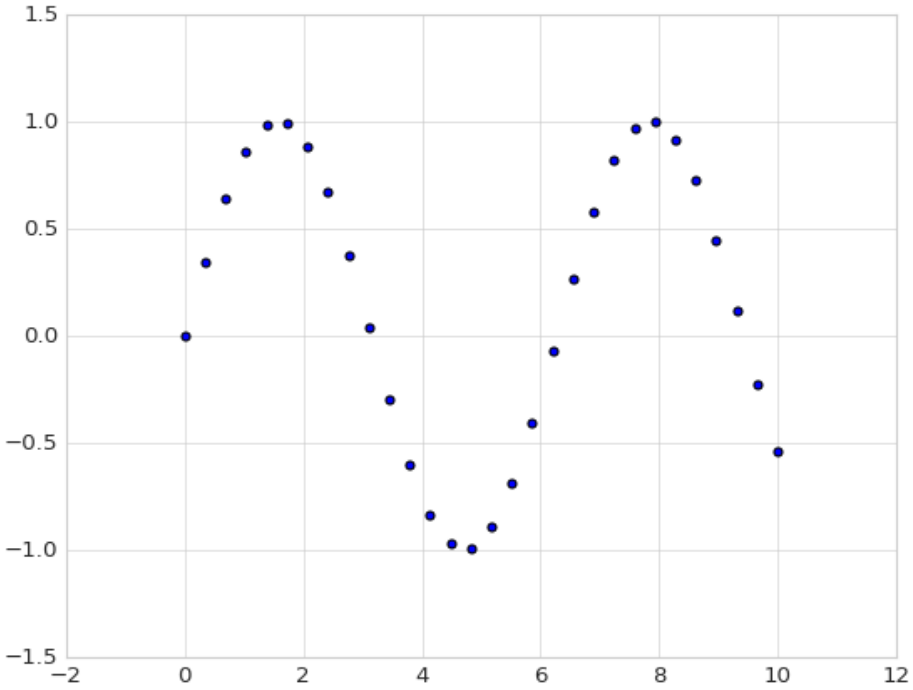
```
import numpy as np

x = np.linspace(0, 10, 30)
y = np.sin(x)
z = np.cos(x)

plt.plot(x, y, '-ok') # line (-), circle marker (0), black (k)
plt.plot(x, z, '-o', color='blue')
plt.show()
```



```
import numpy as np

x = np.linspace(0, 10, 30)
y = np.sin(x)
z = np.cos(x)

plt.plot(x, y, '-ok') # line (-), circle marker (0), black (k)
plt.plot(x, z, '-pb',
         markersize=15, linewidth=4,
         markerfacecolor='white',
         markeredgecolor='gray',
         markeredgewidth=2)

plt.show()
```

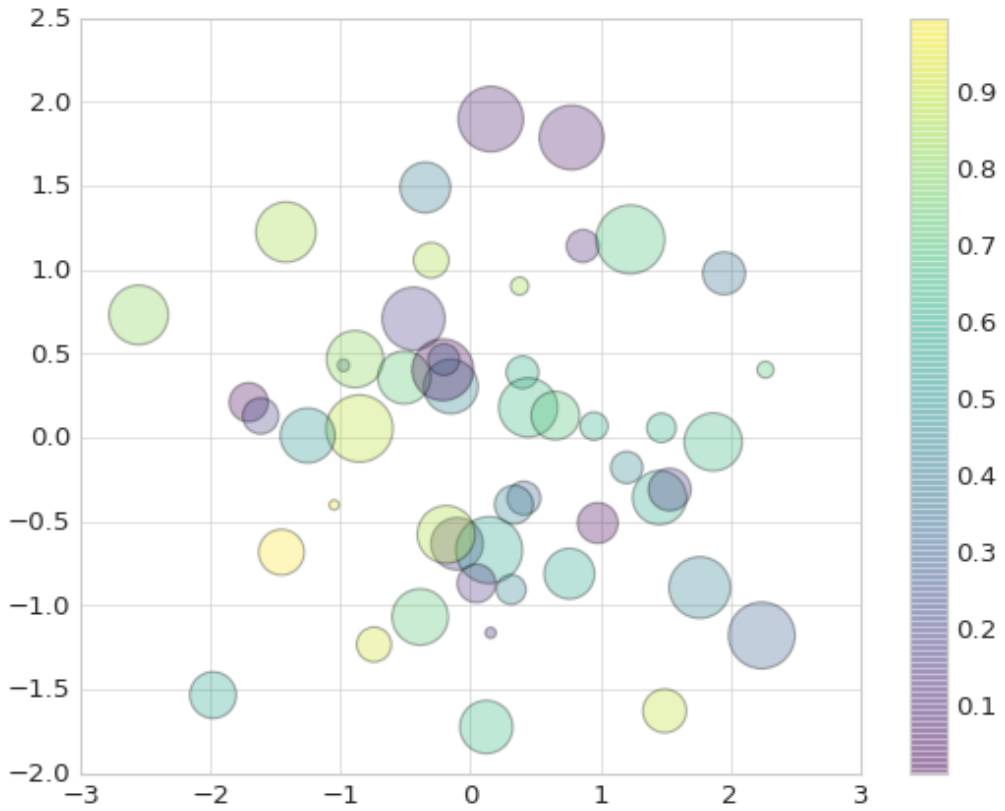


```
x = np.linspace(0, 10, 30)
y = np.sin(x)

plt.scatter(x, y, marker='o')
plt.show()
```

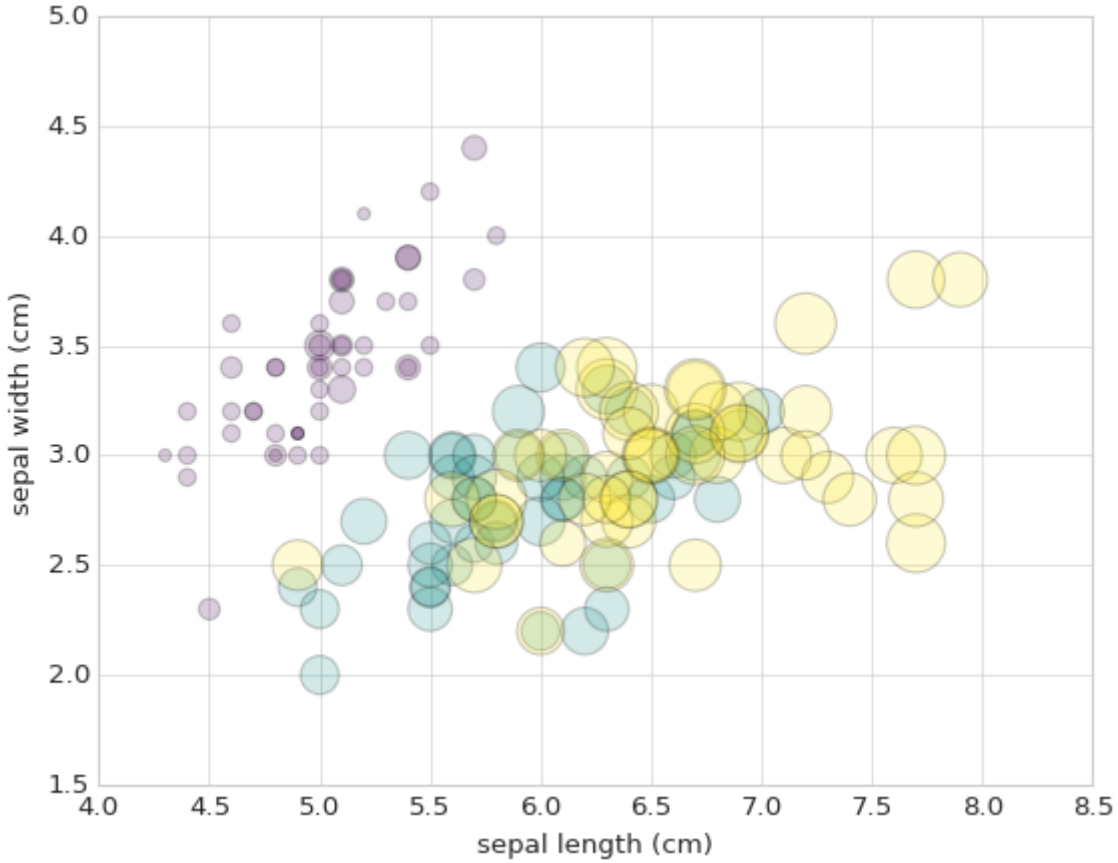


การตกแต่ง Scatter

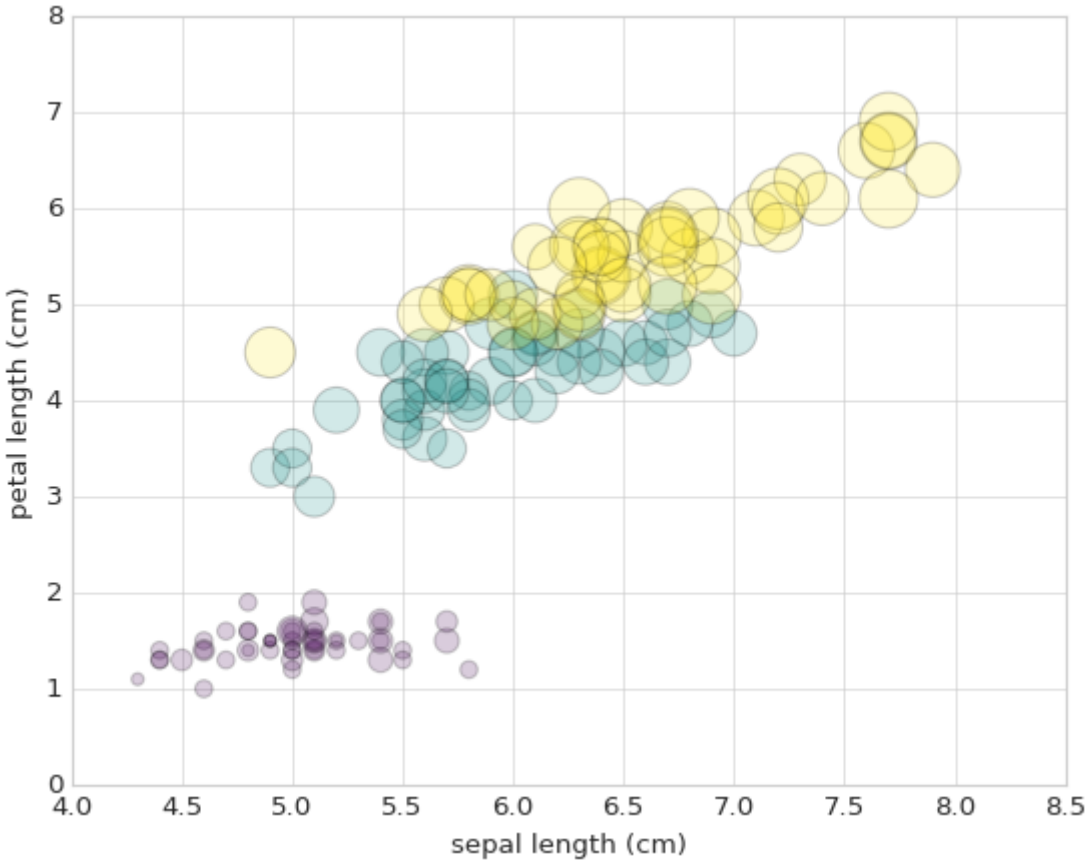
```
rng = np.random.RandomState(0)

no_plot = 50
x = rng.randn(no_plot)
y = rng.randn(no_plot)
colors = rng.rand(no_plot)
sizes = 1000 * rng.rand(no_plot)

plt.scatter(x, y, c=colors, s=sizes, alpha=0.3,
                cmap='viridis')
plt.colorbar()
plt.show()
```



```
from sklearn.datasets import load_iris

iris_dataset = load_iris()
features = iris_dataset.data.T   # T = Matrix Transpose

plt.scatter(features[0], features[1], alpha=0.2,
      s=300*features[3], c=iris_dataset.target,
      cmap='viridis')

plt.xlabel(iris_dataset.feature_names[0])
plt.ylabel(iris_dataset.feature_names[1])
plt.show()
```

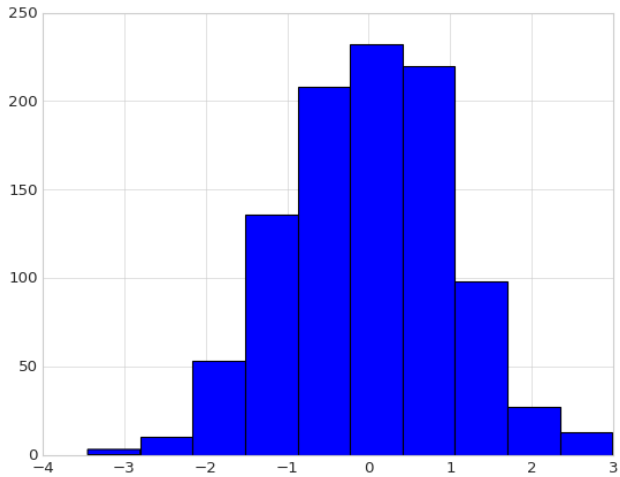


```
from sklearn.datasets import load_iris

iris_dataset = load_iris()
features = iris_dataset.data.T   # T = Matrix Transpose

plt.scatter(features[0], features[2], alpha=0.2,
           s=300*features[3], c=iris_dataset.target,
           cmap='viridis')

plt.xlabel(iris_dataset.feature_names[0])
plt.ylabel(iris_dataset.feature_names[2])
 plt.show()
```

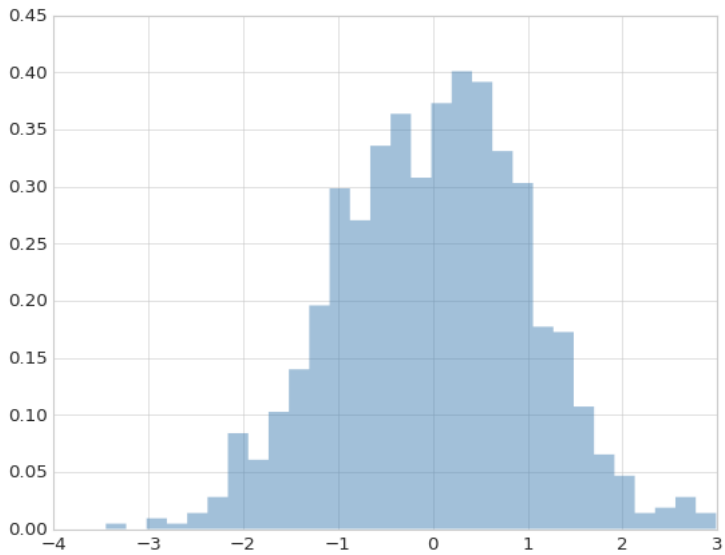


การพลอต Histograms, Binnings และ Density

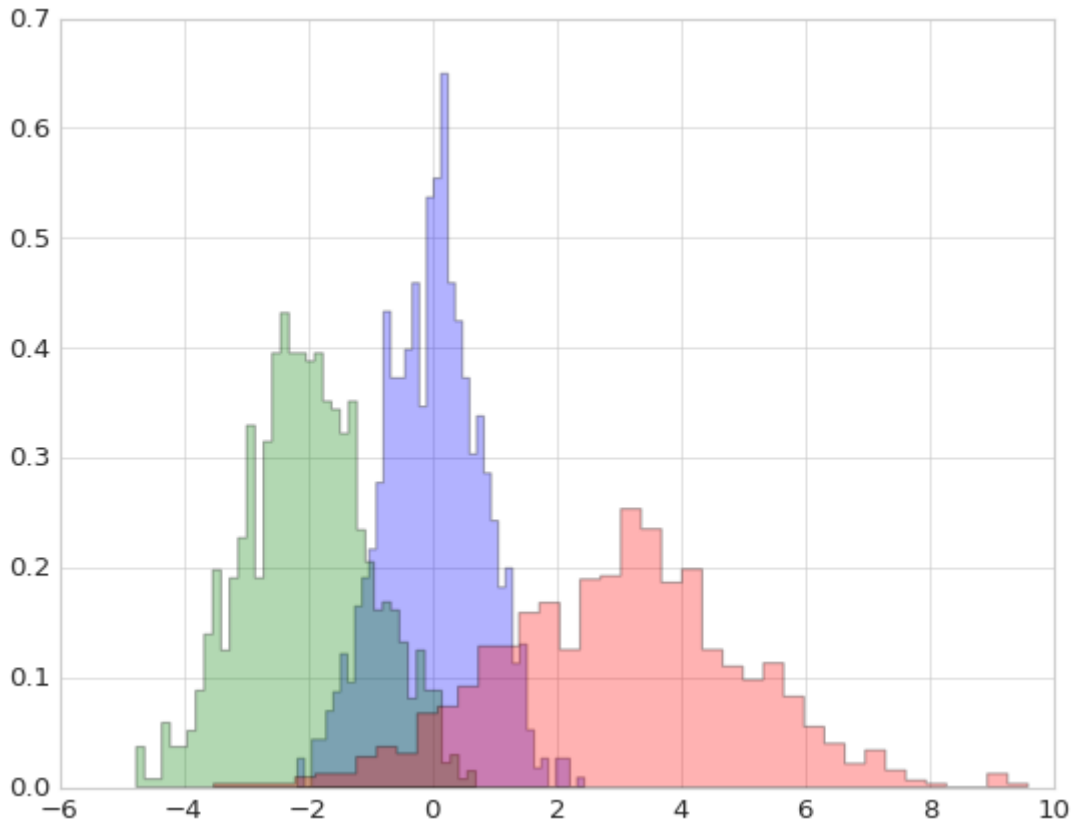
```
import numpy as np
import matplotlib.pyplot as plt

data = np.random.randn(1000)

# simple histogram
plt.hist(data)
plt.show()
```




```
plt.hist(data, bins=30, normed=True, alpha=0.5,
          histtype='stepfilled', color='steelblue',
          edgecolor='none')
plt.show()
```



```
# over-plotting multiple histograms
x1 = np.random.normal(0, 0.8, 1000)
x2 = np.random.normal(-2, 1, 1000)
x3 = np.random.normal(3, 2, 1000)

kwargs = dict(histtype='stepfilled', alpha=0.3,
              normed=True, bins=40)

plt.hist(x1, **kwargs)
plt.hist(x2, **kwargs)
plt.hist(x3, **kwargs)
plt.show()
```



การคำนวณค่า Histogram

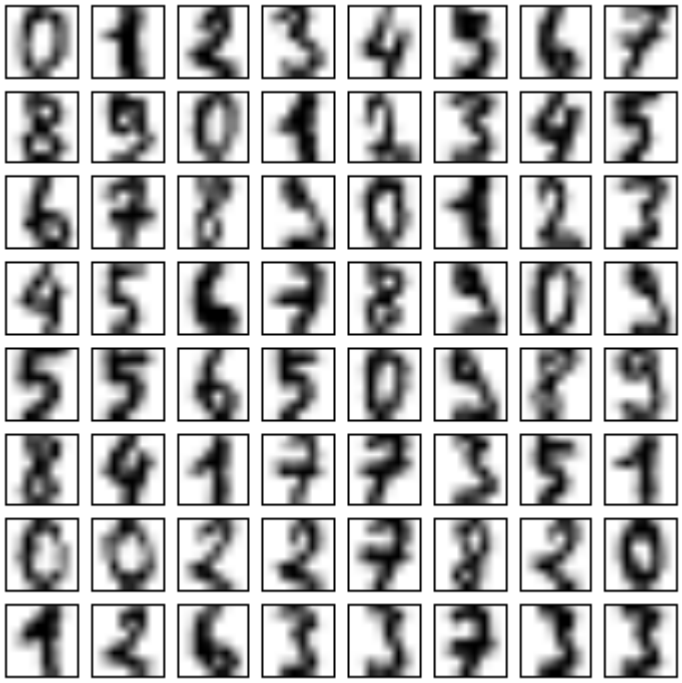
```
counts, bin_edges = np.histogram(data, bins=5)
print(counts)

counts, bin_edges = np.histogram(data, bins=7)
print(counts)
```

```
[ 33 202 534 219  12]
[ 14  65 231 397 224  65   4]
```

การพลอตชุดข้อมูล MNSIT ที่อยู่ใน scikit-learn

การพลอตชุดข้อมูล MNSIT ที่เรียกใช้จากโปรแกรม scikit-learn สามารถทำได้ดังนี้

```
from sklearn import datasets

digits = datasets.load_digits()
fig, ax = plt.subplots(8, 8, figsize=(6,6))
for i, axi in enumerate(ax.flat):
    axi.imshow(digits.images[i], cmap='binary')
    axi.set(xticks=[], yticks=[])
```



การพลอตชุดข้อมูล MNIST ในรูปแบบ 2 มิติ โดยใช้วิธี IsoMap

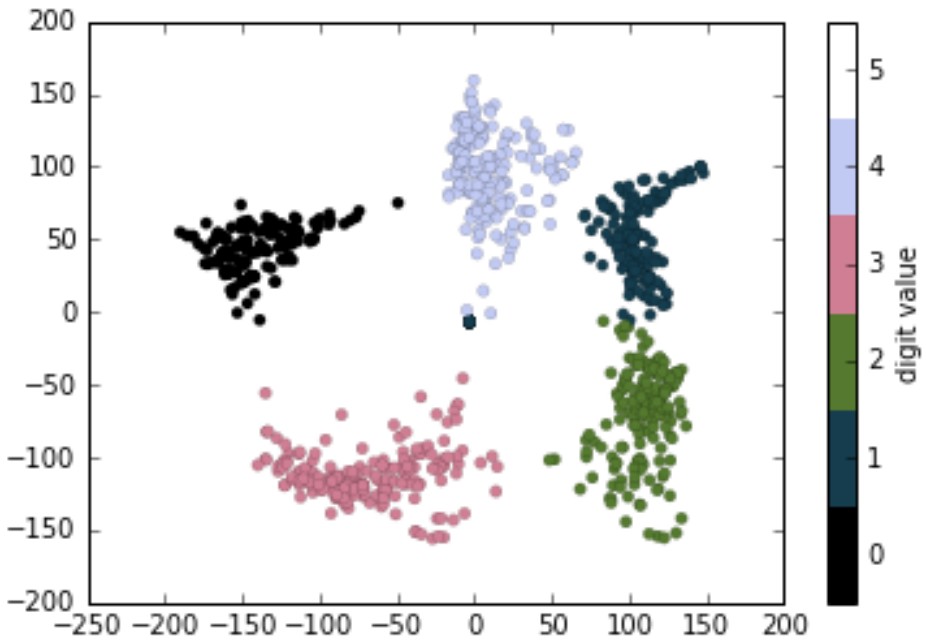
```
from sklearn import datasets
digits = datasets.load_digits(n_class=5)
```

ตัวอย่างข้างต้น แสดงวิธีการดึงข้อมูล MNIST โดยกำหนดให้ n_class=5 นั่นหมายถึงโหลดเฉพาะ class 0-4 เท่านั้น

```
from sklearn.manifold import Isomap

iso = Isomap(n_components=2)
projection = iso.fit_transform(digits.data)
```

จากนั้นเรียกใช้เมธอด Isomap เพื่อคำนวณหาค่าสหสัมพันธ์โดยกำหนดให้ components = 2

```
plt.scatter(projection[:, 0], projection[:, 1], lw=0.1,
               c=digits.target,
               cmap=plt.cm.get_cmap('cubehelix', 6))
plt.colorbar(ticks=range(6), label='digit value')
plt.clim(-0.5, 5.5)

plt.show()
```



บทที่ 5
ไลบรารี่ Seaborn
(Seaborn Library)

Seaborn Library เป็นไลบรารี่ที่ใช้งานร่วมกับ Python เพื่อใช้แสดงข้อมูลทางสถิติ (statistical data visualization) การติดตั้งไลบรารี่ Seaborn ทำได้โดยเปิดโปรแกรม Terminal และพิมพ์คำสั่งดังต่อไปนี้

```
$ pip install seaborn
```

หากต้องการติดตั้งผ่าน Jupyter สามารถทำได้โดย

```
! pip install seaborn
```

ไลบรารี่ seaborn จะทำงานควบคู่กับไลบรารี่ Pandas ดังนั้น หากยังไม่ได้ติดตั้ง Pandas สามารถทำได้โดยใช้คำสั่งดังต่อไปนี้

```
$ pip install pandas
```

เมื่อติดตั้ง seaborn เป็นที่เรียบร้อย สามารถทดสอบโดยอิมพอร์ตไลบรารี่ seaborn มาใช้งาน ดังนี้

```
import seaborn as sns
```

ทดสอบการใช้ไลบรารี่ seaborn โดยเรียกใช้ชุดข้อมูล Iris ทำได้ดังต่อไปนี้

```
import seaborn as sns

iris_dataset = sns.load_dataset('iris')
iris_dataset.head()
```

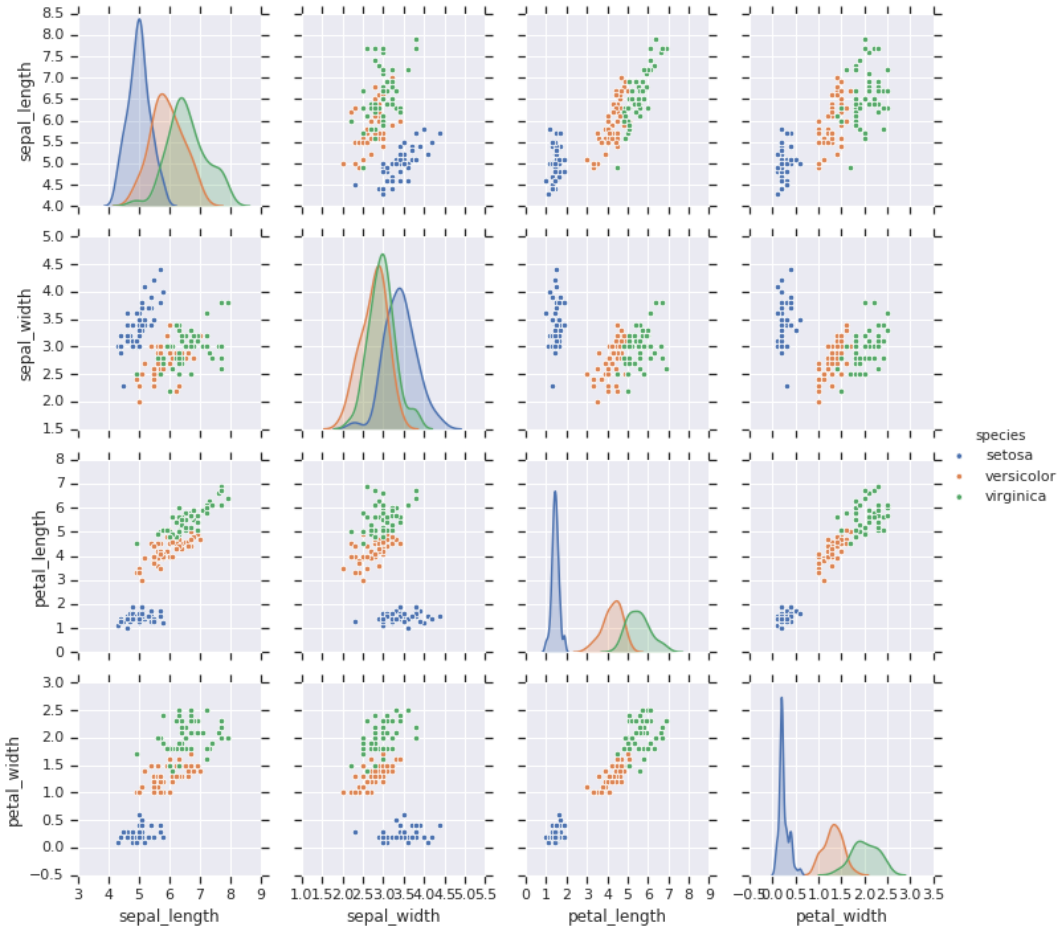
เมื่อโหลดชุดข้อมูล iris เสร็จสิ้น ตัวแปร iris_dataset จะถูกจัดเก็บให้อยู่ในรูปแบบของ pandas DataFrame หากต้องการดูข้อมูลที่อยู่ใน DataFrame สามารถใช้คำสั่ง head() เพื่อดูราย

ละเอียด โดยคำสั่ง head() จะแสดงตัวอย่างจำนวน 5 ตัวอย่างเท่านั้น ผลลัพธ์ที่ได้จากการใช้คำสั่ง iris_dataset.head() แสดงดังต่อไปนี้

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

การแสดงชุดข้อมูล Iris แบบ Visualization

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set()
sns.pairplot(iris_dataset, hue='species', size=2.5)
plt.show()
```



การทำงานร่วมกันระหว่าง seaborn และ scikit-learn

ในกรณีที่ใช้ seaborn โหลดข้อมูลมาใช้งาน ข้อมูลจะถูกจัดเก็บในรูปแบบของ DataFrame ดังนั้นหากต้องการดึงข้อมูล (Extract) อาร์เรย์ feature และ target ออกมาจาก DataFrame สามารถทำได้ดังนี้

```
X_iris = iris_dataset.drop('species', axis=1)
X_iris.shape
```

```
(150, 4)
```

จากตัวอย่างข้างต้น คำสั่ง iris_dataset.drop('species', axis=1) เป็นการลบข้อมูลของ 'species' ในแนวคอลัมน์ ซึ่งมีทั้งหมด 1 attribute ออกจาก iris_dataset

```
y_iris = iris_dataset['species']
y_iris.shape
```

```
(150,)
```

จากตัวอย่างข้างต้น เลือกคอลัมน์ 'species' ซึ่งมีทั้งหมด 1 attribute เพื่อใช้เป็นข้อมูล y หรือ label

หากต้องการดูประเภทของข้อมูลสามารถใช้คำสั่ง type() แสดงดังตัวอย่างต่อไปนี้

```
type(X_iris)
```

```
pandas.core.frame.DataFrame
```

จากตัวอย่างข้างต้นแสดงให้เห็นว่าตัวแปร X_iris เป็นข้อมูลประเภท DataFrame ดังนั้นสามารถใช้คำสั่ง head() เพื่อดูข้อมูลของ X_iris

```
X_iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width
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

หากต้องการเรียกดูข้อมูลที่ละส่วน (Slice) สามารถทำได้ดังต่อไปนี้

```
X_iris.iloc[1]  # row 1
```

```
sepal_length    4.9
sepal_width     3.0
petal_length    1.4
petal_width     0.2
Name: 1, dtype: float64
```

สามารถใช้ index ในการเรียกดูข้อมูล ซึ่งคล้ายกับอาร์เรย์ จากตัวอย่างต่อไปนี้ คือการเรียกดูข้อมูลที่ row = 0 และ col = 0

```
X_iris.iloc[0,0]  # row 0, col 0
5.1
```

หากต้องการเรียกดูข้อมูลในทุก ๆ row ที่ col = 1 สามารถทำได้โดย

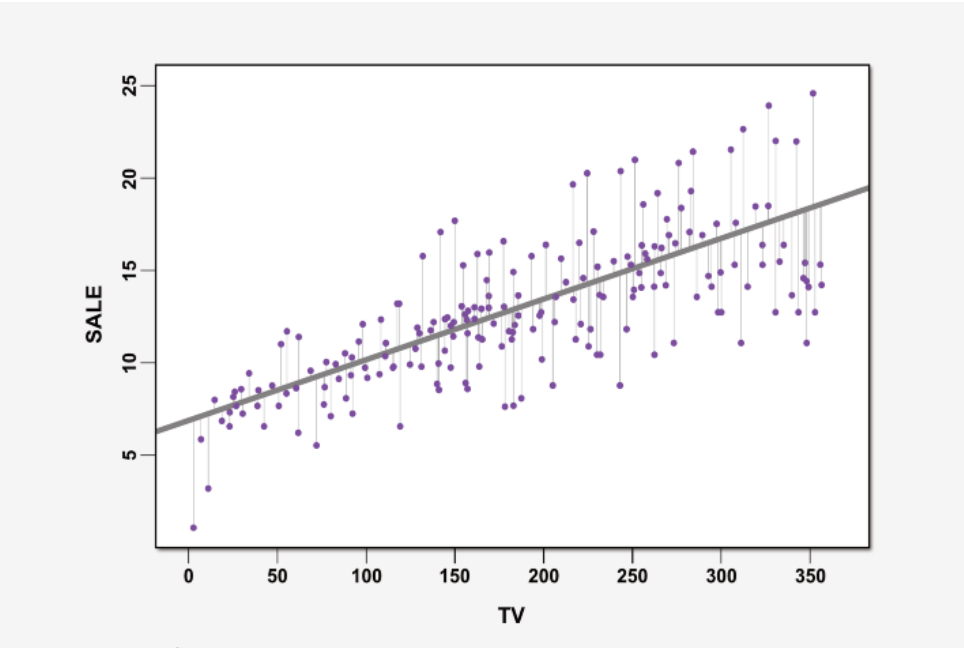
```
X_iris.iloc[:,1]
```

```
0      3.5
1      3.0
2      3.2
3      3.1
4      3.6
5      3.9
      ...
144    3.3
145    3.0
146    2.5
147    3.0
148    3.4
149    3.0
Name: sepal_width, dtype: float64
```

บทที่ 6
การวิเคราะห์การถดถอยเชิงเส้น (Linear Regression)

การวิเคราะห์การถดถอยเชิงเส้น (Linear Regression) เป็นการคำนวณหาความสัมพันธ์ระหว่างตัวแปรตั้งแต่ 2 ตัวแปร โดยเป็นการประมาณการ (Predictor, X) และตัวตอบสนอง (Response, y) ซึ่งเป็นความสัมพันธ์แบบเชิงเส้น (Linear) ซึ่งเป็นการคำนวณจากค่า X และ y ที่มีความสัมพันธ์กัน เพื่อคำนวณออกมาเป็นสมการความสัมพันธ์ สมการของ Linear Regression แสดงดังต่อไปนี้

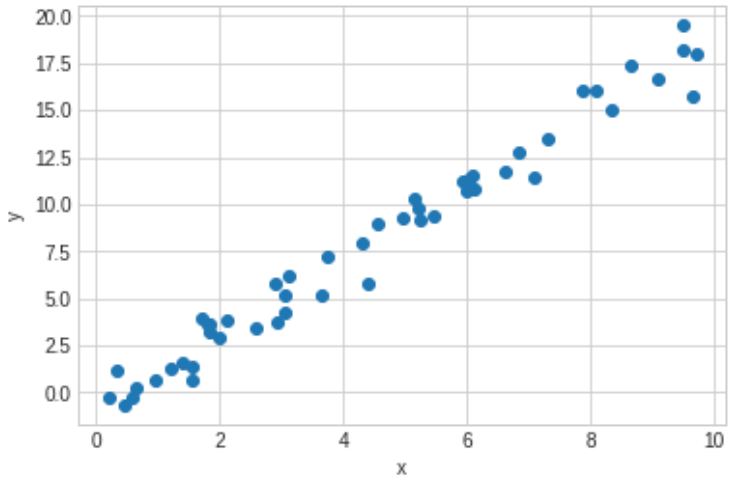
$$y = ax + b$$



ภาพประกอบที่ 9: แสดงสมการถดถอยเชิงเส้น Linear Regression
ที่มา : https://goo.gl/QL6rCo

การจำลองชุดข้อมูลเพื่อใช้ในการคำนวณ Linear Regression

```
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('seaborn-whitegrid')

# create random data
rng = np.random.RandomState(42)
x = 10 * rng.rand(50)
y = 2 * x - 1 + rng.randn(50)

plt.scatter(x,y)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



การเรียกใช้โมดูล LinearRegression

การเรียกใช้ Linear Regression ในโปรแกรม scikit-learn สามารถทำได้ดังนี้

```
from sklearn.linear_model import LinearRegression
```

จากนั้นสามารถกำหนดค่าพารามิเตอร์ (Hyperparameter) เช่น

```
model = LinearRegression(fit_intercept=True)
print(model)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

จัดการข้อมูลเพื่อใช้ในการคำนวณ

```
X = x[:, np.newaxis]
print(X.shape)
```

```
(50, 1)
```

จากตัวอย่างข้างต้นข้อมูลที่จะนำไปใช้ในการคำนวณ Linear Regression กำหนดให้มีขนาดเป็น (50, 1) นั่นหมายถึงมีตัวแปร X เพียงตัวแปรเดียว ซึ่งเรียกว่าการคำนวณแบบ Simple Linear Regression หากมีตัวแปร X มากกว่า 1 ตัวแปรจะเรียกว่า Multiple Linear Regression

```
print X[:10]
```

```
[[ 3.74540119]
 [ 9.50714306]
 [ 7.31993942]
 [ 5.98658484]
 [ 1.5601864 ]
 [ 1.5599452 ]
 [ 0.58083612]
 [ 8.66176146]
 [ 6.01115012]
 [ 7.08072578]]
```

หากต้องการแสดงข้อมูลของตัวแปร X สามารถทำได้โดยใช้คำสั่ง print และสามารถ Slice ข้อมูลออกมาดูโดยกำหนด index ที่ต้องการเช่น X[:10] หมายถึง การแสดงข้อมูล X ตั้งแต่ ข้อมูลลำดับที่ 0 ถึง 10

การเรียนรู้เพื่อสร้างโมเดล (Train the model)

```
model.fit(X,y)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

ในโปรแกรม scikit-learn สามารถใช้คำสั่ง fit เพื่อที่จะสร้างโมเดลขึ้นมาใช้งาน ดังตัวอย่างข้างต้น

การพยากรณ์ผลลัพธ์จากข้อมูลใหม่ (Predict Labels for Unknown Data)

เนื่องจากตัวอย่างข้างต้นได้จำลองข้อมูลเพื่อนำมาใช้ในการเรียนรู้ของ Linear Regression ดังนั้นในส่วนของการพยากรณ์ข้อมูลจึงทำการจำลองข้อมูลชุดใหม่อีก 1 ชุด เพื่อใช้สำหรับการพยากรณ์ การจำลองข้อมูลสามารถทำได้ดังนี้

```
xfit = np.linspace(-1, 11)

# Xfit = unknown data
Xfit = xfit[:, np.newaxis]

print(Xfit.shape)
print(Xfit[:10])
```

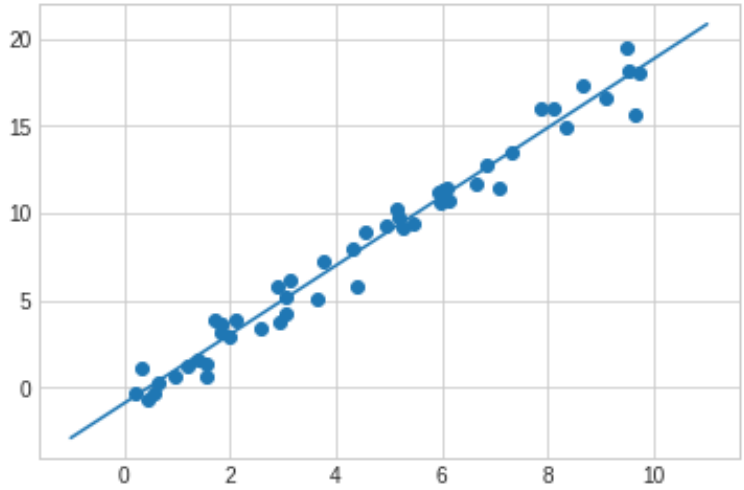
```
(50, 1)

[[-1.        ]
 [-0.75510204]
 [-0.51020408]
 [-0.26530612]
 [-0.02040816]
 [ 0.2244898 ]
 [ 0.46938776]
 [ 0.71428571]
 [ 0.95918367]
 [ 1.20408163]]
```

ตัวอย่างข้างต้นได้จำลองข้อมูลจำนวน 50 จุดเพื่อใช้สำหรับการพยากรณ์ จากนั้นนำข้อมูลที่ได้ไปพยากรณ์ด้วยคำสั่ง predict ดังตัวอย่างต่อไปนี้

```
yfit = model.predict(Xfit)
```

ผลการพยากรณ์จะถูกเก็บไว้ในตัวแปร yfit ดังนั้นสามารถนำผลลัพธ์ที่ได้มาพลอตกราฟเพื่อแสดงเส้น Linear ที่ได้จากการพยากรณ์ ดังตัวอย่างต่อไปนี้

```
plt.scatter(x, y)
plt.plot(xfit, yfit)
plt.show()
```



ภาพประกอบที่ 10: เส้น Hyperplane ที่ได้จากการคำนวณด้วยวิธี Linear Regression

การพยากรณ์ข้อมูล Diabetes ด้วย Linear Regression

อิมพอร์ตไลบรารี่ที่จำเป็นสำหรับการทำ Linear Regression

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
```

โหลดชุดข้อมูล Diabetes

```
diabetes = datasets.load_diabetes()
```

ตรวจสอบขนาดของข้อมูล

```
print("shape of the dataset", diabetes.data.shape)
```

```
('shape of the dataset', (442, 10))
```

ตรวจสอบข้อมูลตั้งแต่ row ลำดับที่ 0 ถึง 5 และ column ลำดับที่ 0 ถึง 5

```
diabetes.data[0:5,0:5]
```

```
array([[ 0.03807591,  0.05068012,  0.06169621,  0.02187235, -0.0442235 ],
       [-0.00188202, -0.04464164, -0.05147406, -0.02632783, -0.00844872],
       [ 0.08529891,  0.05068012,  0.04445121, -0.00567061, -0.04559945],
       [-0.08906294, -0.04464164, -0.01159501, -0.03665645,  0.01219057],
       [ 0.00538306, -0.04464164, -0.03638469,  0.02187235,  0.00393485]])
```

ในกรณีนี้ค่า X จะถูกกำหนดให้ใช้ข้อมูล diabetes ใน column ที่ 2

```
diabetes_X = diabetes.data[:, np.newaxis, 2]  # coloumn 2
diabetes_X[:10]
```

```
array([[ 0.06169621],
       [-0.05147406],
       [ 0.04445121],
       [-0.01159501],
       [-0.03638469],
       [-0.04069594],
       [-0.04716281],
       [-0.00189471],
       [ 0.06169621],
       [ 0.03906215]])
```

เนื่องจากข้อมูล diabetes มีจำนวน 442 instance และไม่ได้แบ่งข้อมูลออกเป็นชุดทดสอบและชุดเรียนรู้ จึงต้องแบ่งข้อมูลออกเป็นสองส่วน

```
# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]
```

ในกรณีนี้ไม่ได้ใช้ฟังก์ชัน train_test_split เพื่อแบ่งข้อมูล แต่ได้กำหนดให้ข้อมูลตั้งแต่ row ลำดับที่ 0 ถึงลำดับที่ 422 เป็นข้อมูลสำหรับการเรียนรู้ และลำดับที่ 423 ถึง 442 เป็นข้อมูลสำหรับการทดสอบ ดังนั้น [:-20] จึงหมายถึงข้อมูลตั้งแต่ลำดับแรกไปจนถึงลำดับสุดท้าย แต่ลบด้วย 20 (442-20 = 422)

ขั้นตอนต่อไป ทำการสร้างโมเดลด้วยคำสั่ง LinearRegression() และเรียนรู้ (Train) ด้วยข้อมูลชุดเรียนรู้ ซึ่งก็คือ diabetes_X_train

```
regr = linear_model.LinearRegression()
regr.fit(diabetes_X_train, diabetes_y_train)
```

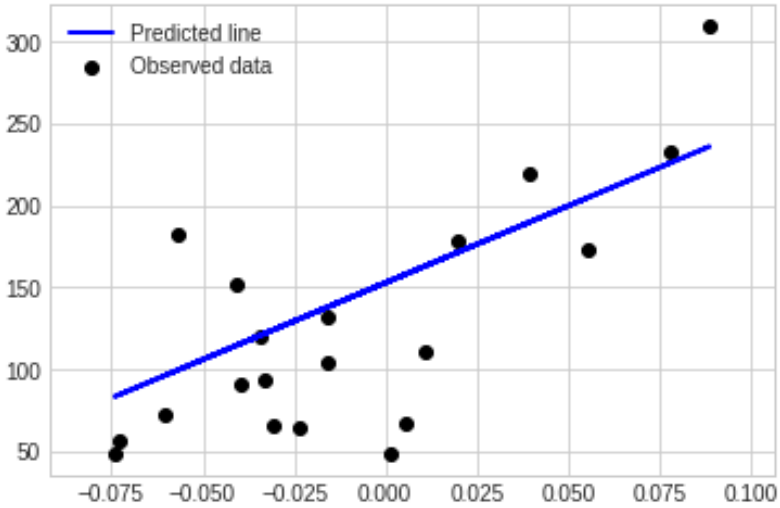
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1,
normalize=False)
```

จากนั้น ทำการพยากรณ์ข้อมูลชุดทดสอบ diabetes_X_test ด้วยคำสั่ง predict()

```
diabetes_y_pred = regr.predict(diabetes_X_test)
```

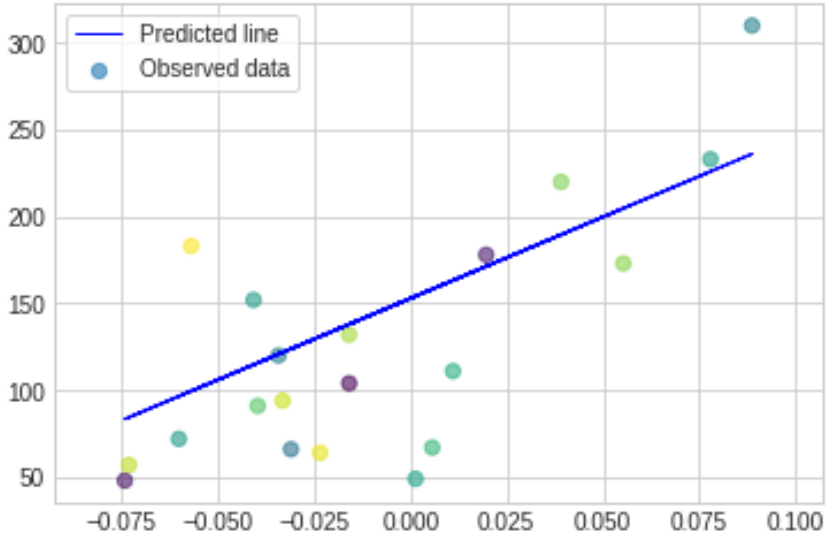
ทำการพลอตกราฟเพื่อดูข้อมูล และเส้น Hyperplane ที่ได้จากการคำนวณ

```
import matplotlib.pyplot as plt

plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue',
          linewidth=2)
plt.legend(['Predicted line', 'Observed data'])

plt.show()
```



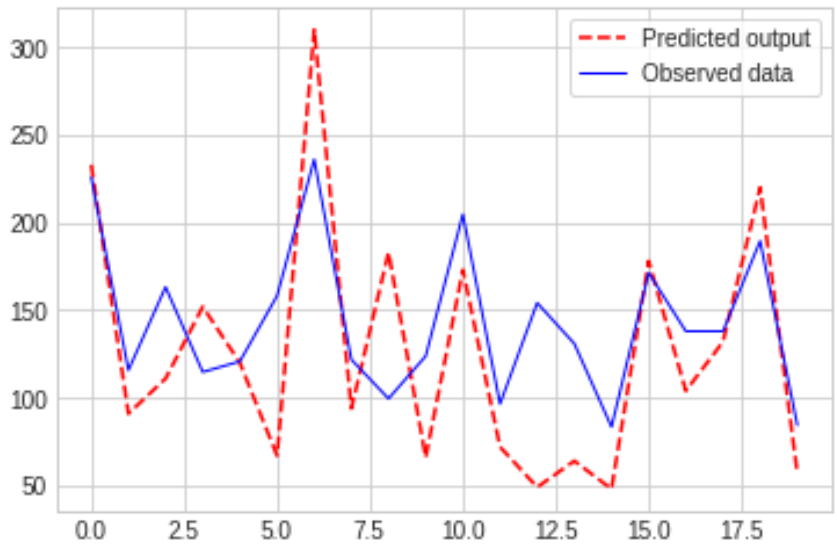
ปรับแต่งกราฟให้สวยงาม ทำได้ดังนี้

```
import numpy as np
import matplotlib.pyplot as plt

rng = np.random.RandomState(0)
colors = rng.rand(20)

fig = plt.figure()
plt.scatter(diabetes_X_test, diabetes_y_test,  c=colors, s=40,
               alpha=0.6, cmap='viridis')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue',
               linewidth=1)
plt.legend(['Predicted line', 'Observed data'], frameon=True,
               loc='upper left')

plt.show()
```



หากต้องการเปรียบเทียบระหว่าง actual target และ predict target สามารถทำได้ ดังนี้

```
plt.plot(diabetes_y_test, '--r')
plt.plot(diabetes_y_pred, '-b', linewidth=1)
plt.legend(['Predicted output', 'Observed data'],
                frameon=True, loc='upper right')

plt.show()
```



ภาพประกอบที่ 11: แสดงการพลอตค่า target จริง (Actual Target) และค่า target ที่ได้จากการพยากรณ์ (Predicted Target)

สามารถพิมพ์ค่า coefficients และค่า Mean Squared Error (MSE) ออกมาเพื่อตรวจสอบได้ดังนี้

```
# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(diabetes_y_test,
     diabetes_y_pred))
```

```
('Coefficients: \n', array([ 938.23786125]))
Mean squared error: 2548.07
Variance score: 0.47
```

การพยากรณ์ข้อมูล Housing ด้วย Linear Regression

อิมพอร์ตไลบรารี่ที่จำเป็นสำหรับการทำ Linear Regression

```
import pandas as pd
import numpy as np
from sklearn import linear_model
from sklearn.cross_validation import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

โหลดข้อมูล boston housing ที่อยู่ใน scikit-learn มาใช้งาน

```
from sklearn.datasets import load_boston  # housing dataset
boston = load_boston()
```

ตัวแปร boston ถูกจัดเก็บอยู่ในรูปแบบ dictionary สามารถเรียกดูรายละเอียดของการจัดเก็บข้อมูลด้วยคำสั่ง keys()

```
print(boston.keys())
```

```
['data', 'feature_names', 'DESCR', 'target']
```

หากต้องการดูรายละเอียดของชุดข้อมูล Boston housing สามารถทำได้โดย

```
print(boston.DESCR)  หรือคำสั่ง print(boston['DESCR'])
```

```
Boston House Prices dataset
===========================

Notes
------
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000
sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0
otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
```

```
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by
town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None
```

หากต้องการทราบขนาดของชุดข้อมูล Boston housing สามารถทำได้โดย

```
print("shape of the data", boston.data.shape)
#print("shape of the data", boston['data'].shape)
boston['data'][:2,:] # row = 2, col = all
```

```
('shape of the data', (506, 13))

array([[  6.32000000e-03,   1.80000000e+01,   2.31000000e+00,
          0.00000000e+00,   5.38000000e-01,   6.57500000e+00,
          6.52000000e+01,   4.09000000e+00,   1.00000000e+00,
          2.96000000e+02,   1.53000000e+01,   3.96900000e+02,
          4.98000000e+00],
       [  2.73100000e-02,   0.00000000e+00,   7.07000000e+00,
          0.00000000e+00,   4.69000000e-01,   6.42100000e+00,
          7.89000000e+01,   4.96710000e+00,   2.00000000e+00,
          2.42000000e+02,   1.78000000e+01,   3.96900000e+02,
          9.14000000e+00]])
```

ข้อมูลชุด Boston housing มีจำนวน 506 instance และมี feature ทั้งสิ้น 13 attribute หากต้องการดู Label ของข้อมูลสามารถทำได้โดยใช้คำสั่ง boston[‘target’] และใช้คำสั่ง shape เพื่อดูขนาดของข้อมูล

```
print("shape of the target", boston['target'].shape)
boston['target'][:10]
```

```
('shape of the target', (506,))
array([ 24. ,  21.6,  34.7,  33.4,  36.2,  28.7,  22.9,  27.1,  16.5,  18.9])
```

เพื่อช่วยในการแสดงข้อมูลสามารถแปลงข้อมูลให้อยู่ในรูปแบบของ pandas DataFrame ดังต่อไปนี้

```
df_x = pd.DataFrame(boston.data, columns=boston.feature_names)
df_y = pd.DataFrame(boston.target)
```

สามารถดูรายละเอียดของข้อมูลโดยเรียกผ่าน DataFrame ดังต่อไปนี้

```
df_x.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.593761	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407
std	8.596783	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000
75%	3.647423	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000

ภาพประกอบที่ 12: ตัวอย่างข้อมูลที่จัดเก็บในรูปแบบของ pandas DataFrame

```
df_y.describe()
```

	0
count	506.000000
mean	22.532806
std	9.197104
min	5.000000
25%	17.025000
50%	21.200000
75%	25.000000
max	50.000000

เมื่อเตรียมข้อมูลเรียบร้อยแล้ว จากนั้นทำการสร้างโมเดลของ Linear Regression

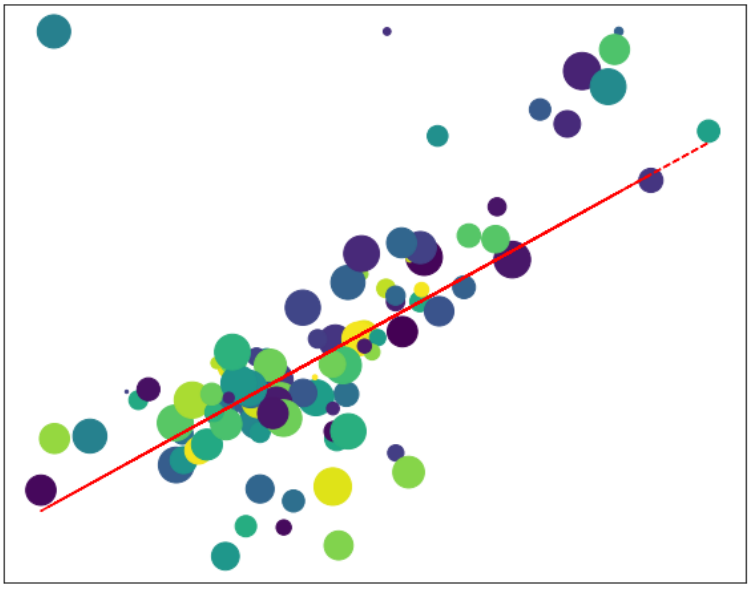
```
import pandas as pd
import numpy as np
from sklearn import linear_model
from sklearn.cross_validation import train_test_split
from sklearn.datasets import load_boston  # housing dataset

boston = load_boston()
```

ในกรณีนี้จะเลือกใช้เพียง 1 attribute/feature เท่านั้นเพื่อสร้าง Model ในกรณีนี้เลือกใช้ข้อมูลจาก column ลำดับที่ 5 คือ RM

```
# select column 5 = RM
y = boston['target']
X = boston['data']
x_train, x_test, y_train, y_test = train_test_split(X[:,5], y,
test_size=0.2, random_state=4)

# create a model
reg.fit(x_train.reshape(-1, 1), y_train.reshape(-1, 1))

# prediction
y_pred = reg.predict(x_test.reshape(-1, 1))
```

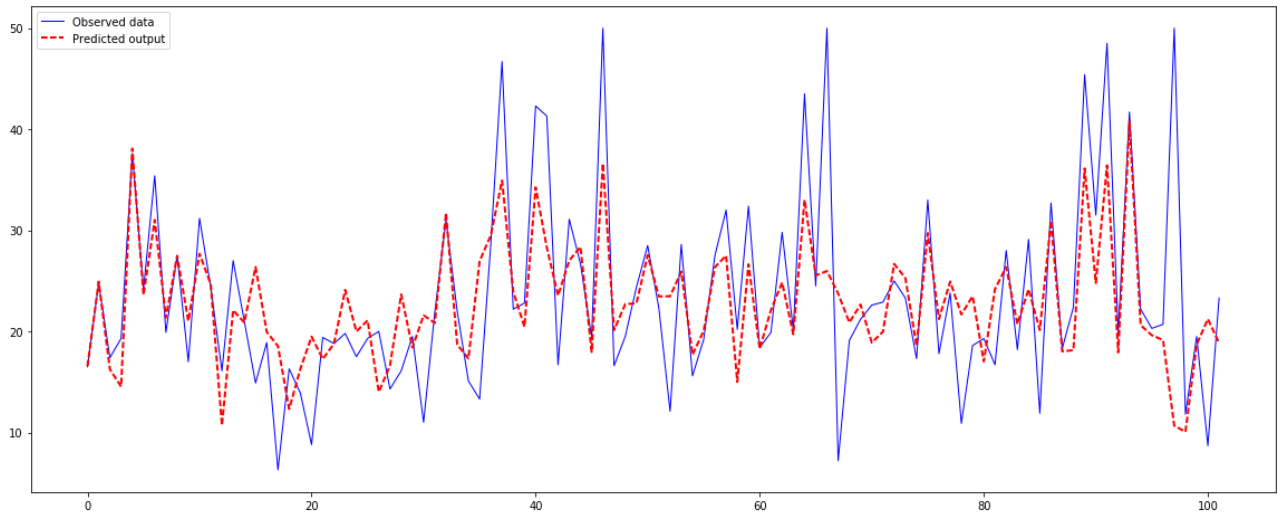
คำสั่ง fit() คือการสร้างโมเดล และ predict() คือการพยากรณ์ ผลลัพธ์ที่ได้จากการพยากรณ์ถูกเก็บไว้ในตัวแปร y_pred

```
import numpy as np
rng = np.random.RandomState(42)
colors = rng.rand(x_test.shape[0])
sizes = 800 * rng.rand(x_test.shape[0])

plt.figure(figsize=(10,8))
plt.scatter(x_test, y_test,  c=colors, s=sizes)
plt.plot(x_test, y_pred, '--r')

plt.xticks(())
plt.yticks(())

plt.show()
```



หากต้องการเปรียบเทียบระหว่างค่าจริง (Actual) และค่าที่พยากรณ์ (Predicted) สามารถสร้างกราฟเพื่อแสดงข้อมูลในลักษณะของ Visualization ได้ดังนี้

```
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 8))
plt.plot(y_test, linewidth=1, linestyle='solid',
          color='blue', label='Observed data')
plt.plot(y_pred, '--r', linewidth=2, label='Predicted output')
plt.legend(frameon=True, loc='upper left')

plt.show()
```



แสดงค่า Coefficients, MSE และ Variance score

```
# The coefficients
print('Coefficients: \n', reg.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(y_test, y_pred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, y_pred))
```

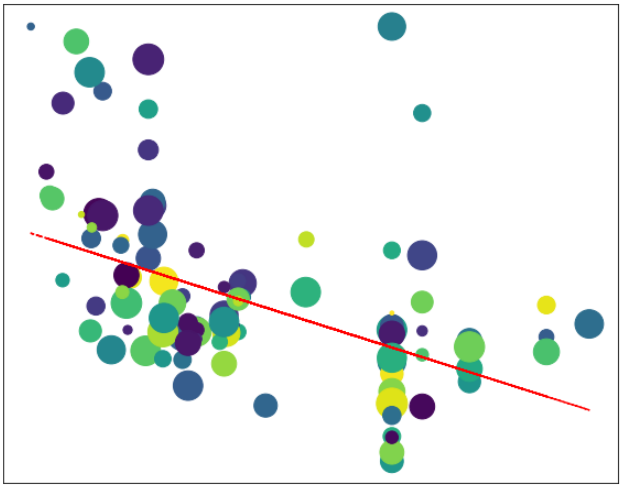
```
('Coefficients: \n', array([[ -1.14743504e-01,   4.70875035e-02,   8.70282354e-03,
          3.23818824e+00,  -1.67240567e+01,   3.87662996e+00,
         -1.08218769e-02,  -1.54144627e+00,   2.92604151e-01,
         -1.33989537e-02,  -9.07306805e-01,   8.91271054e-03,
         -4.58747039e-01]]))
Mean squared error: 25.41
Variance score: 0.73
```

ในกรณีต่อไปจะใช้ column ลำดับที่ 2 คือ INDUS ในการสร้างโมเดล

```
X = boston['data']
x_train, x_test, y_train, y_test =
train_test_split(X[:,2], y, test_size=0.2,
                    random_state=4)

# create a model
reg.fit(x_train.reshape(-1, 1), y_train.reshape(-1, 1))

# prediction
y_pred = reg.predict(x_test.reshape(-1, 1))
```

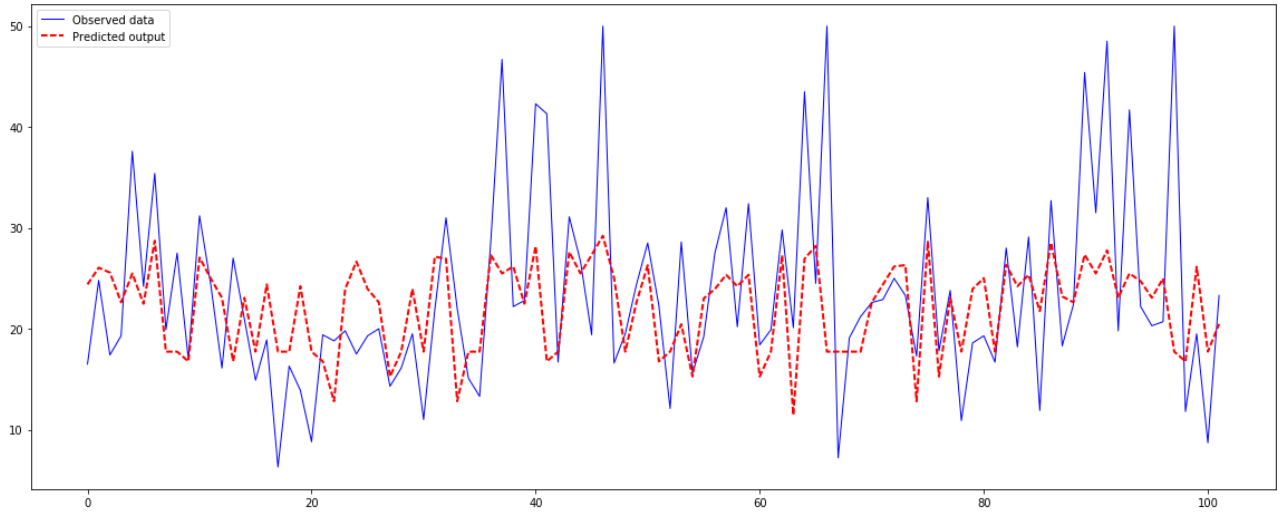
จากนั้นพลอตกราฟ เพื่อแสดงในลักษณะของการ Visualization

```
import numpy as np
import matplotlib.pyplot as plt

# Plot outputs
rng = np.random.RandomState(42)
colors = rng.rand(x_test.shape[0])
sizes = 800 * rng.rand(x_test.shape[0])

plt.figure(figsize=(10,8))
plt.scatter(x_test, y_test,  c=colors, s=sizes)
plt.plot(x_test, y_pred, '--r')

plt.xticks(())
plt.yticks(())

plt.show()
```



```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(20, 8))
plt.plot(y_test, linewidth=1, linestyle='solid', color='blue',
          label='Observed data')
plt.plot(y_pred, '--r', linewidth=2, label='Predicted output')
plt.legend(frameon=True, loc='upper left')

plt.show()
```




```
# The coefficients
print('Coefficients: \n', reg.coef_)
# The mean squared error
print("Mean squared error: %.2f"
     % mean_squared_error(y_test, y_pred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, y_pred))
```

```
('Coefficients: \n', array([[-0.65143285]]))
Mean squared error: 75.47
Variance score: 0.19
```

สรุปผลการทดลองด้วยวิธี Liner Regression กับข้อมูลชุด Boston housing

จากการทดลองทั้ง 2 กรณี คือใช้คอลัมน์ RM และ INDUS ปรากฏว่า

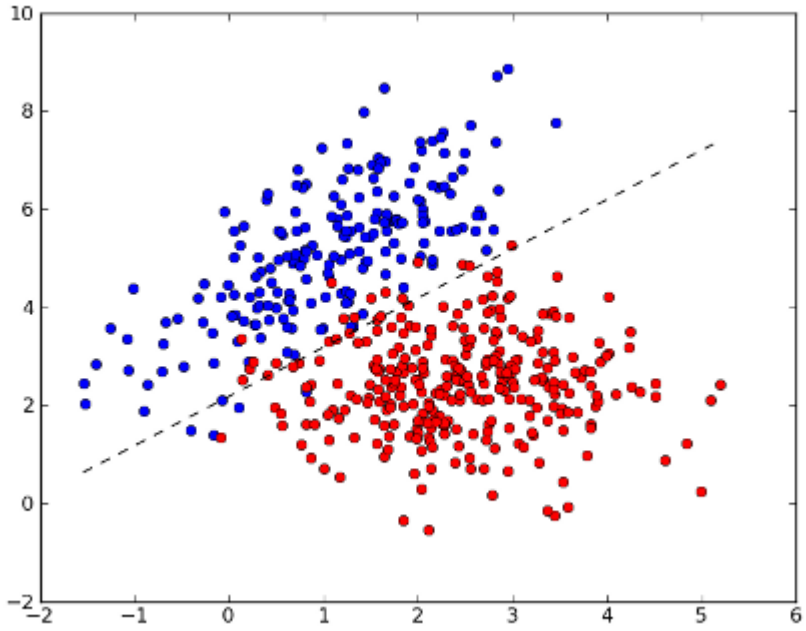
RM มีค่า MSE 25.41 และ Variance score 0.73

INDUS มีค่า MSE 75.47 และ Variance score 0.19

โดยที่ค่า MSE หากเข้าใกล้ 0 จะมีค่า Error น้อยที่สุด และค่า Variance score หากมีค่าเป็น 1 จะเป็นการพยากรณ์ที่แม่นยำที่สุด จึงสรุปได้ว่า หากใช้คอลัมน์ RM และ INDUS ในการพยากรณ์ข้อมูล Boston housing คอลัมน์ RM มีประสิทธิภาพในการพยากรณ์สูงกว่า ดังนั้น หากต้องการนำไปใช้ในการสร้างโมเดล จึงแนะนำให้ใช้คอลัมน์ RM ในการสร้าง ทั้งนี้ อาจจะต้องลองทำการสร้างโมเดลด้วยวิธีอื่น เช่น Multiple Linear Regression เพื่อทดสอบประสิทธิภาพต่อไป

บทที่ 7
ตัวจำแนกแบบไบนารี่
(Binary Classifier)

ตัวจำแนกแบบไบนารี่ (Binary Classifier) เป็นวิธีการแบ่งกลุ่มข้อมูลออกเป็นสองกลุ่ม (Binary Class) จากตัวอย่างข้อมูลแบ่งออกเป็นสองกลุ่ม (สีน้ำเงิน และสีแดง) โดยตัวจำแนกแบบไบนารีจะสร้างเส้นสมมุติหรือเรียกว่า hyperplane เพื่อใช้สำหรับการแบ่งกลุ่มข้อมูลทั้งสองกลุ่มออกจากกัน



ภาพประกอบที่ 13: แสดงเส้น Hyperplane ที่ใช้แบ่งข้อมูลออกเป็นสองส่วน

ในกรณีนี้จะทดสอบการจำแนกแบบไบนารี โดยใช้ข้อมูล MNSIT ในการทดสอบ

```
from scipy.io import loadmat

mnist_raw = loadmat("mldata/mnist-original.mat")
mnist = {
    "data": mnist_raw["data"].T,
    "target": mnist_raw["label"][0],
    "COL_NAMES": ["label", "data"],
    "DESCR": "mldata.org dataset: mnist-original",
    }

X,y = mnist['data'], mnist['target']
```

จากตัวอย่างข้างต้น ใช้โปรแกรม scipy ในการโหลดไฟล์นามสกุล .mat ซึ่งบรรจุด้วยชุดข้อมูล MNSIT โดยข้อมูลสามารถดาวน์โหลดได้จาก

https://github.com/amplab/datascience-sp14/blob/master/lab7/mldata/mnist-original.mat

เมื่อโหลดข้อมูลเสร็จเรียบร้อยสามารถเรียกดูรายละเอียดของข้อมูลได้ดังนี้

```
# 70K images, 28x28 pixels/image, each pixel = 0 (white) to 255 (black)
mnist # a dict object
```

```
{'COL_NAMES': ['label', 'data'],
 'DESCR': 'mldata.org dataset: mnist-original',
 'data': array([[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]], dtype=uint8),
 'target': array([0., 0., 0., ..., 9., 9., 9.])}
```

ข้อมูลของตัวเลขจะถูกจัดเก็บอยู่ในรูปแบบของเวกเตอร์ ที่มีขนาด 754 attribute (28x28) ดังนั้น จึงกำหนดให้ตัวแปร X เก็บข้อมูล Feature และ y เก็บ label ที่มีจำนวนทั้งสิ้น 70,000 ตัว

```
X,y = mnist['data'], mnist['target']
X.shape, y.shape
```

```
((70000, 784), (70000,))
```

จากนั้นทำการแยกชุดข้อมูลออกเป็น 2 ชุด คือ Training set และ Test set

```
import numpy as np

X_train, X_test, y_train, y_test = X[:60000], X[60000:], \
          y[:60000], y[60000:]

shuffle_index = np.random.permutation(60000)
X_train, y_train = X_train[shuffle_index], \
          y_train[shuffle_index]

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

```
((60000, 784), (10000, 784), (60000,), (10000,))
```

ชุดข้อมูล MNIST ที่โหลดมาได้แบ่งข้อมูลเป็นที่เรียบร้อย โดยข้อมูลชุดที่ 1 – 60000 เป็นชุดเรียนรู้ โดยข้อมูลจะเรียงจาก กลุ่ม (Class) 0 ถึง Class 9 และตั้งแต่ 60001 – 70000 เป็นชุดทดสอบ ข้อมูลจะเรียงลำดับจาก Class 0 ถึง Class 9 เช่นกัน ดังนั้น สามารถใช้คำสั่งต่อไปนี้สำหรับแบ่งข้อมูล

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], \
          y[:60000], y[60000:]
```

Stochastic Descent

เริ่มต้นด้วยการกำหนดข้อมูลให้เป็น 2 กลุ่ม (Class) โดยตัวอย่างกำหนดให้กลุ่มของตัวเลข 5 มีค่าเป็น True และกลุ่มตัวเลขตั้งแต่ 0-4 และ 6-9 มีค่าเป็น False

```
y_train_5 = (y_train == 5) # create target vectors
y_test_5  = (y_test == 5)

print(y_train_5.shape, y_train_5)
print(y_test_5.shape, y_test_5)
```

```
((60000,), array([False, False, False, ..., False, False, False]))
((10000,), array([False, False, False, ..., False, False, False]))
```

สร้างโมเดล Stochastic Descent

สำหรับ scikit-learn ให้เรียกใช้คำสั่ง SGDClassifier() เพื่อกำหนด hyperparameter ที่จำเป็น และใช้คำสั่ง fit() เพื่อสร้างโมเดล

```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

```
SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', n_iter=5, n_jobs=1,
       penalty='l2', power_t=0.5, random_state=42, shuffle=True, verbose=0,
       warm_start=False)
```

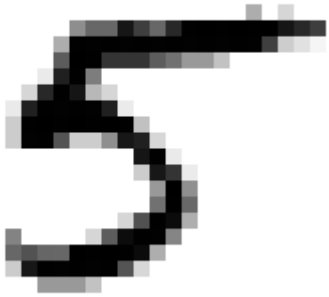
การพยากรณ์ด้วยโมเดล Stochastic Descent

สำหรับโปรแกรม scikit-learn สามารถใช้คำสั่ง predict() สำหรับการพยากรณ์ข้อมูล ในกรณีนี้ได้สร้างฟังก์ชัน (Function) ในภาษา Python สามารถประกาศฟังก์ชันด้วยคำสั่ง def ประกอบด้วย plot_digit() และ pred_data() เพื่อให้เรียกใช้งานได้อย่างสะดวก

```
def plot_digit(X):
    plt.imshow(
    X.reshape(28, 28),
    cmap = plt.cm.binary,
    interpolation="nearest")

    plt.axis("off")
    plt.show()

def print_pred_data(clf, actual_y, X):
    print("Actual : ", actual_y)
    print("Prediction : ", clf.predict([X])[0])
```

ดังนั้น การพยากรณ์ข้อมูลตัวเลข และแสดงรูปภาพของตัวเลขแบบ Visualization สามารถทำได้ดังตัวอย่างต่อไปนี้

```
pred_digit = 5500
plot_digit(X_test[pred_digit])
print_pred_data(sgd_clf, y_test_5[pred_digit], \
                X_test[pred_digit])
```



```
('Actual : ', True)
('Prediction : ', True)
```

จากตัวอย่างข้างต้น ได้พยากรณ์ข้อมูลในชุดทดสอบลำดับที่ 5500 (X_test[5500]) ซึ่ง Class ที่แท้จริง (Actual Class) คือ True (ตัวเลข 5) และผลการพยากรณ์ (Prediction Class) ได้คำตอบคือ True นั่นแสดงว่าเป็นการพยากรณ์ที่ถูกต้อง

```
pred_digit = 1000
plot_digit(X_test[pred_digit])
print_pred_data(sgd_clf, y_test_5[pred_digit], \
                X_test[pred_digit])
```

```
('Actual : ', False)
('Prediction : ', False)
```

การวัดประสิทธิภาพ (Performance Measurement)

สำหรับการวัดประสิทธิภาพสามารถทำได้หลายวิธี ในตัวอย่างนี้จะใช้คำสั่ง cross_val_score() เพื่อทดสอบและวัดประสิทธิภาพของอัลกอริธึม Stochastic Descent แสดงดังตัวอย่างต่อไปนี้

```
from sklearn.model_selection import cross_val_score

print(cross_val_score(
        sgd_clf,
        X_train,
        y_train_5,
        cv=3,
        scoring="accuracy"))
```

```
[0.9617  0.96315 0.95905]
```

ตัวอย่างข้างต้นใช้คำสั่ง cross_val_score() ในการวัดประสิทธิภาพ โดยค่ามาตรฐาน (Default) ของคำสั่งสำหรับการวัดค่าความถูกต้องได้กำหนดให้ k-fold มีค่าเท่ากับ 3 (cross validation: cv=3) ดังนั้น ผลลัพธ์ที่ได้ก็คือ 0.9617, 0.96315 และ 0.95905 โดยเป็นผลลัพธ์ของการทดลองครั้งที่ 1, 2 และ 3 ตามลำดับ

การประเมินประสิทธิภาพของอัลกอริธึมด้วย Confusion Matrix

การประเมินประสิทธิภาพของกัลกอริธึมด้วย Confusion Matrix เป็นวิธีที่แสดงให้เห็นถึงผลลัพธ์จากการพยากรณ์ และทำให้รู้ได้ว่าหากผลลัพธ์ผิดพลาด ผิดพลาด ณ จุดใด การทำ Confusion Matrix ใน scikit-learn ใช้คำสั่ง confusion_matrix() สามารถเรียกใช้ได้ดังต่อไปนี้

```
from sklearn.metrics import confusion_matrix
```

ในกรณีนี้ได้สร้างฟังก์ชันเพื่อใช้สำหรับการแสดงผลของ Confusion Matrix แบบ Visualization โดยสร้างฟังก์ชันชื่อ plot_confusion_matrix() แสดงดังต่อไปนี้

```
import matplotlib.pyplot as plt
import itertools

def plot_confusion_matrix(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), \
                                  range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], 'd'),
                 horizontalalignment='center',
                 color='white' if cm[i,j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

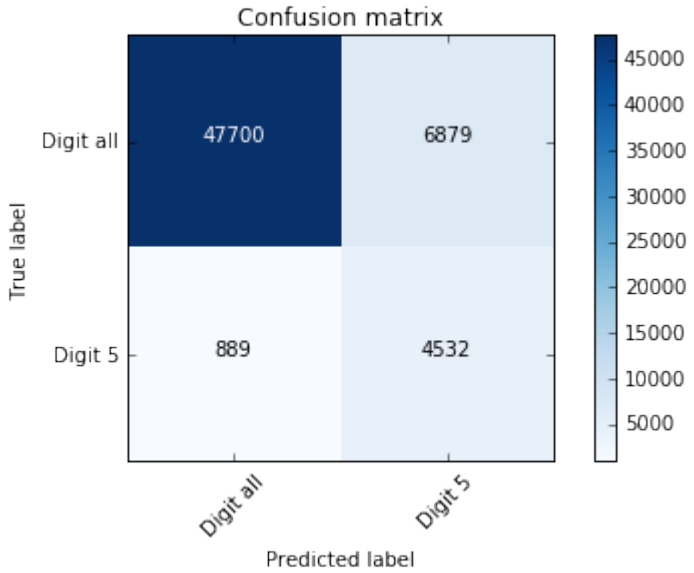
ในกรณีนี้ได้ทดสอบโมเดลของ Stochastic Descent อีกครั้งโดยใช้คำสั่ง cross_val_predict() วิธีการทดสอบ แสดงดังต่อไปนี้

```
from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(sgd_clf, X_train, \
                                 y_train_5, cv=3)
```

จากตัวอย่าง ทดสอบโดยใช้ข้อมูลชุดทดสอบ (Train) เมื่อเสร็จสิ้นการทดสอบผลลัพธ์ที่ได้จากการพยากรณ์จะถูกจัดเก็บที่ตัวแปร y_train_pred จากนั้นจึงนำผลลัพธ์จากการพยากรณ์ (Prediction Class) y_train_pred และผลลัพธ์ที่แท้จริง (Actual Class) ไปคำนวณเพื่อหา Confusion Matrix ดังตัวอย่างต่อไปนี้

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_train_5, y_train_pred)
print(cm)

plt.figure()
classes = ['Digit all', 'Digit 5']
plot_confusion_matrix(cm, classes)
```

```
[[47700  6879]
 [  889  4532]]
```



จากผลลัพธ์ข้างต้น แบ่งออกเป็น 2 แถว และ 2 คอลัมน์ เนื่องจากมีทั้งสิ้น 2 กลุ่ม ข้อมูลที่แสดงในแนวแถวแสดงถึงกลุ่มที่ถูกต้อง (True label) และข้อมูลที่แสดงในแถวคอลัมน์ แสดงถึงผลลัพธ์จากการพยากรณ์ ตัวอย่างเช่น ในแถวข้อมูลชุด Digit all มีการพยากรณ์ถูกต้องจำนวน 47700 และพยากรณ์ผิดเป็น Digit 5 อยู่จำนวน 6879 ตัวเลข และในแถวข้อมูลชุด Digit 5 มีการพยากรณ์ถูกต้องจำนวน 4532 และพยากรณ์ผิดเป็น Digit all จำนวน 889 ตัวเลข

จากนั้นจึงนำผลลัพธ์ที่ได้ y_train_pred ไปคำนวณร่วมกับ y_train_5 เพื่อคำนวณหาค่า Precision, Recall และ F1 ดังต่อไปนี้

```
from sklearn.metrics import precision_score, recall_score, \
                            f1_score

print("precision:",precision_score(y_train_5, y_train_pred))
print("recall:",recall_score(y_train_5, y_train_pred))
print("f1:",f1_score(y_train_5, y_train_pred))
```

```
('precision:', 0.3971606344755061)
('recall:', 0.8360081165836561)
('f1:', 0.5384980988593155)
```

สำหรับการทดสอบโดยใช้ Training set นั้นจะเป็นการทดสอบเพื่อปรับค่าพารามิเตอร์ (Tuning Parameter) ให้เหมาะสมกับชุดข้อมูล และเพื่อให้มีประสิทธิภาพสูงสุด จากนั้นจึงนำโมเดลที่ได้ไปทดสอบกับข้อมูลชุด Test set หรือนำไปใช้งานจริงต่อไป

การนำโมเดลไปทดสอบกับข้อมูลชุดทดสอบ

จากตัวอย่างก่อนหน้านี้ ใช้คำสั่ง precision_score, recall_score และ fa_score เพื่อแสดงผลลัพธ์ ทั้งนี้ สามารถใช้คำสั่ง classification_report() เพื่อแสดงผลลัพธ์ได้เช่นเดียวกัน แสดงดังตัวอย่างต่อไปนี้

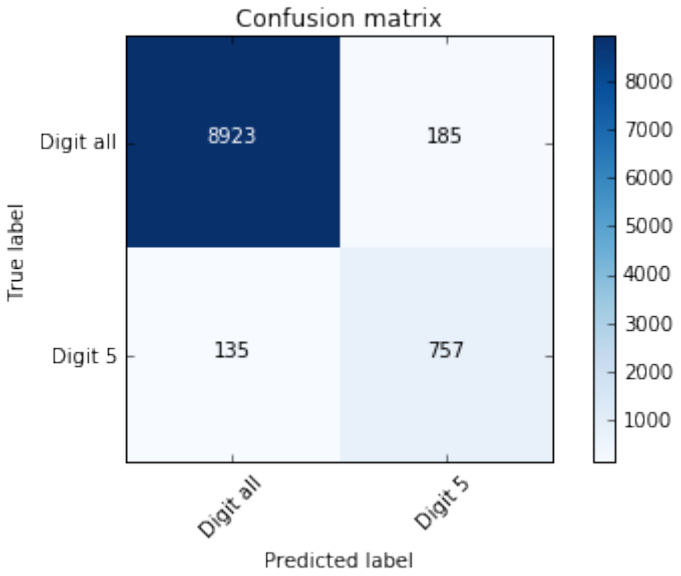
```
from sklearn.metrics import classification_report

y_test_pred = sgd_clf.predict(X_test)

classes = ['Digit all', 'Digit 5']
print(classification_report(y_test_5, y_test_pred, \
                            target_names=classes))
```

```
             precision    recall  f1-score   support

  Digit all       0.99      0.98      0.98      9108
    Digit 5       0.80      0.85      0.83       892

avg / total       0.97      0.97      0.97     10000
```

จากนั้นใช้เรียกใช้ฟังก์ชัน plot_confusion_matrix() เพื่อทำการ Visualization ข้อมูล

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test_5, y_test_pred)
print(cm)

plt.figure()
classes = ['Digit all', 'Digit 5']
plot_confusion_matrix(cm, classes)
```

```
[[8923  185]
 [ 135  757]]
```



เมื่อได้ทำการพยากรณ์ Test set จากนั้นจึงตรวจสอบประสิทธิภาพของอัลกอริธึมโดยใช้คำสั่ง accuracy_score() ดังตัวอย่างต่อไปนี้

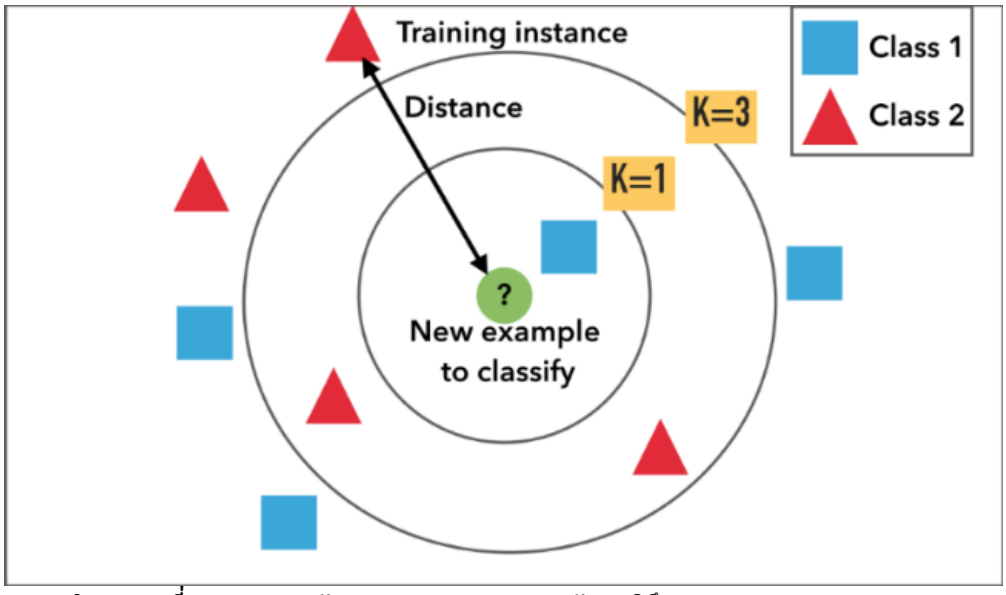
```
from sklearn.metrics import accuracy_score

print("Accuracy Score", accuracy_score(y_test_5, \
                                       y_test_pred)*100)
```

```
('Accuracy Score', 96.8)
```

ผลลัพธ์ที่ได้จากการพยากรณ์ด้วยอัลกอริธึม Stochastic Descent มีความถูกต้อง (Accuracy Score/ Accuracy Result) 96.8%

บทที่ 8
การคำนวณเพื่อนบ้านใกล้ที่สุด k ตำแหน่ง (K-Nearest Neighbors)

การคำนวณเพื่อนบ้านใกล้ที่สุด k ตำแหน่ง (K-Nearest Neighbors) เป็นวิธีการเรียนรู้เครื่องจักรที่ใช้สำหรับจัดหมวดหมู่ข้อมูล (Classification) และเป็นวิธีที่ไม่ซ้ำซ้อน



ภาพประกอบที่ 14: แสดงลักษณะการทำงานของอัลกอริธึม KNN

อัลกอริธึมจะทำการเปรียบเทียบจุดใหม่ (New point) กับจุด (Point) ทั้งหมดที่อยู่ใน Training set เพื่อหาจุดที่ใกล้เคียงกับจุดใหม่ที่สุด โดยกำหนดจำนวนจุดที่ใกล้เคียงกับจุดใหม่จำนวน K จุด เช่น หากกำหนดให้ K=3 ดังนั้น จุดที่ใกล้เคียงกับจุดใหม่ทั้งสิ้นจำนวน 3 จุดจะถูกนำมาพิจารณา จากนั้นทำการกำหนด Label/Class ให้กับจุดใหม่ โดยตรวจสอบกับ Label ของทั้ง 3 จุด หากพบว่ามี Label ไหนมากที่สุด (Majority Vote) ก็จะกำหนดเป็น Label ให้กับจุดใหม่นั้น

จากตัวอย่าง

หากกำหนดให้ **K=1** จุดที่ใกล้กับจุดใหม่ที่สุดคือ Class สี่เหลี่ยม ดังนั้น จุดใหม่นั้นจะถูกกำหนดให้มี Class เป็น สี่เหลี่ยม

หากกำหนดให้ **K=3** จุดที่ใกล้กับจุดใหม่ที่สุดสามลำดับคือ Class สี่เหลี่ยม 1 จุด และสามเหลี่ยม 2 จุด ดังนั้นจุดใหม่จะถูกกำหนดให้มี Class เป็น สามเหลี่ยม

การสร้างโมเดลของ KNN

ตัวอย่างดังต่อไปนี้ทำการสร้างโมเดลของอัลกอริธึม KNN โดยใช้ชุดข้อมูล Iris ในการทดสอบ

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris_dataset = load_iris()
X_train, X_test, y_train, y_test =
train_test_split(iris_dataset['data'], \
                         iris_dataset['target'], \
                         random_state=0)
```

จากตัวอย่างข้างต้น ทำการโหลดชุดข้อมูล Iris มาเพื่อทดสอบ โดยใช้คำสั่ง load_iris() จากนั้นใช้คำสั่ง train_test_split() เพื่อแบ่งข้อมูลออกเป็น 2 ชุด โดยข้อมูล Feature/Attribute ที่แบ่งแล้วจะถูกจัดเก็บไว้ที่ตัวแปร X_train และ X_test และข้อมูล Label/Class จะถูกเก็บไว้ที่ตัวแปร y_train และ y_test

จากนั้นใช้คำสั่ง KNeighborsClassifier() เพื่อกำหนดค่าพารามิเตอร์ n_neighbors หากไม่กำหนดค่าโปรแกรมจะใช้ค่าเริ่มต้น (Default) ซึ่งกำหนดให้มีค่าเท่ากับ 5

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=1)
```

จากตัวอย่างข้างต้นได้กำหนดให้ n_neighbors=1 โดยเก็บค่าที่กำหนดไว้ที่ตัวแปร knn จากนั้นทำการสร้างโมเดลโดยใช้คำสั่ง fit() ดังตัวอย่างต่อไปนี้

```
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

การพยากรณ์โดยใช้โมเดลของ KNN

ในชุดข้อมูล Iris นั้นจะมี Label อยู่จำนวน 3 กลุ่ม ประกอบด้วย

setosa กลุ่ม 0
versicolor กลุ่ม 1
virginica กลุ่ม 2

ดังนั้น Label นี้จะใช้เป็นตัวตรวจสอบความถูกต้องของการพยากรณ์

```
print("Test data: {}".format(X_test[1]))
print("Label: {}".format(y_test[1]))
```

```
Test data: [6.  2.2 4.  1. ]
Label: 1
```

ตัวอย่างข้างต้นแสดงให้เห็นข้อมูลที่จะนำไปทดสอบ (Test) การพยากรณ์ด้วยโมเดลของ KNN

```
prediction = knn.predict([X_test[1]])
print("Prediction: {}".format(prediction))
print("Predicted target name: {}".format(        \
          iris_dataset['target_names'][prediction]))
```

```
Prediction: [1]
Predicted target name: ['versicolor']
```

สำหรับการพยากรณ์จะใช้คำสั่ง predict() จากตัวอย่างได้ส่งข้อมูล X_test[1] เข้าไปคำนวณ และผลลัพธ์ที่ได้จะจัดเก็บไว้ที่ตัวแปร prediction จากการทดลองพบว่า เมื่อส่งข้อมูล X_test[1] เข้าไปทดสอบ ผลลัพธ์ที่ได้คือ Prediction: [1] คือกลุ่ม 1 ซึ่งกลุ่ม 1 ก็คือ versicolor ซึ่งเป็นคำตอบที่ถูกต้อง

ทำการทดสอบข้อมูล x_test[2] เพื่อตรวจสอบความถูกต้อง

```
prediction = knn.predict([X_test[2])

print("Prediction: {}".format(prediction))
print("Predicted target name: {}".format( \
               iris_dataset['target_names'][prediction]))
```

```
Prediction: [0]
Predicted target name: ['setosa']
```

จากการทดสอบผลลัพธ์ที่ได้คือกลุ่ม 0 ซึ่งก็คือกลุ่ม setosa

การทดสอบประสิทธิภาพของโมเดล KNN

การแสดงผลลัพธ์ของการจัดกลุ่มข้อมูล (Classification Report) สามารถทำได้ดังตัวอย่างต่อไปนี้

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred,
target_names=iris_dataset['target_names']))
```

```
             precision    recall  f1-score   support

     setosa       1.00      1.00      1.00        13
 versicolor       1.00      0.94      0.97        16
  virginica       0.90      1.00      0.95         9

avg / total       0.98      0.97      0.97        38
```

```
from sklearn.metrics import accuracy_score

accuracy_score(y_test, y_pred)
```

```
0.9736842105263158
```

ในการแสดงผลลัพธ์จากการทดลองในโปรแกรม scikit-learn สามารถทำได้โดยใช้คำสั่ง เช่น classification_report() และ accuracy_score() เป็นต้น จากการทดสอบโมเดลของ KNN มีประสิทธิภาพที่ 97.37%

การแสดงผลการทดลองด้วย Confusion Matrix

```
from sklearn.metrics import confusion_matrix

print("Confusion Matrix: \n{}".format(confusion_matrix( \
                                        y_test, \ y_pred)))
```

```
Confusion Matrix:
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
```

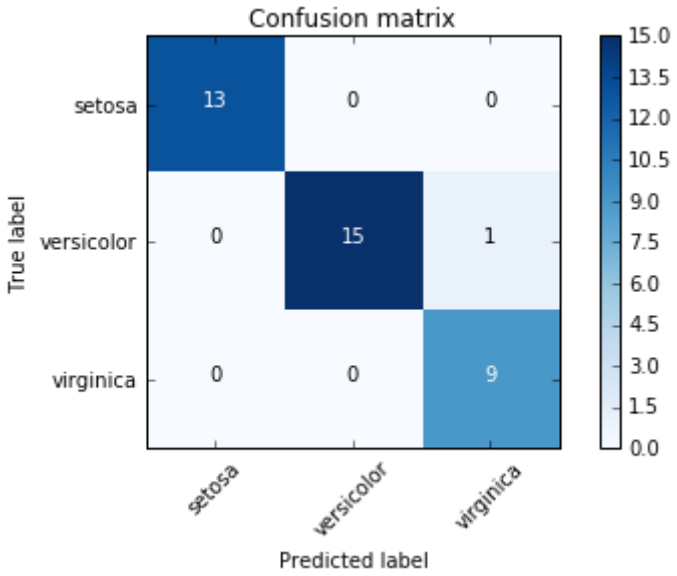
ในการแสดงผล Confusion Matrix ให้อยู่ในรูปแบบของการ Visualization สามารถทำได้โดยการสร้างฟังก์ชัน plot_confusion_matrix() ดังตัวอย่างต่อไปนี้

```
import matplotlib.pyplot as plt
import itertools

def plot_confusion_matrix(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), \
                                  range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], 'd'),
                 horizontalalignment='center',
                 color='white' if cm[i,j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

ในการคำนวณหาค่า Confusion Matrix สามารถใช้คำสั่ง confusion_matrix() ดังตัวอย่างต่อไปนี้

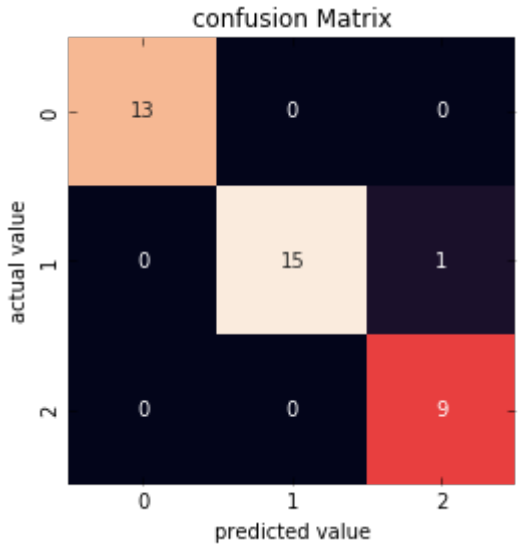
```
cm = confusion_matrix(y_test, y_pred)
np.set_printoptions(precision=2)

print('Confusion matrix, without normalization')
print(cm)

plt.figure()
classes = iris_dataset.target_names
plot_confusion_matrix(cm, classes)
```

```
Confusion matrix, without normalization
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
```



ในการแสดง Confusion Matrix นอกจากจะใช้โปรแกรม matplotlib ยังสามารถใช้โปรแกรม Seaborn เป็นอีกหนึ่งทางเลือก แสดงดังต่อไปนี้

```
import seaborn as sns

sns.heatmap(cm, square=True, annot=True, cbar=False)
plt.title('confusion Matrix')
plt.xlabel('predicted value')
plt.ylabel('actual value')
plt.show()
```



ตัวอย่าง Classification Score และ Confusion Matrix ข้างต้นได้มาจากการกำหนดให้ n_neighbors = 1 เพื่อทดสอบประสิทธิภาพ สามารถกำหนดจำนวนของ n_neighbors ให้เป็น 3, 5 หรือ 7 เป็นต้น

การใช้งานอัลกอริธึม KNN กับข้อมูลโรคเบาหวาน (Diabetes Dataset)

ตัวอย่างต่อไปจะนำอัลกอริธึม KNN มาช่วยในการพยากรณ์ข้อมูลโรคเบาหวาน (Diabetes) ว่าผู้ป่วย (patient) เป็นโรคเบาหวาน หรือไม่ ข้อมูลโรคเบาหวานสามารถดาวน์โหลดได้จากเว็บไซต์

https://www.kaggle.com/amolbhivarkar/knn-for-classification-using-scikit-learn/data

ตัวอย่างการนำอัลกอริธึม KNN ไปใช้พยากรณ์ข้อมูลผู้ป่วยโรคเบาหวาน สามารถทำได้ดังต่อไปนี้

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

plt.style.use('ggplot')

df = pd.read_csv('mldata/diabetes.csv')
df.head()
```

จากตัวอย่างข้างต้น ไฟล์ที่ดาวน์โหลดจากเว็บไซต์อยู่ในรูปแบบของ csv จึงใช้โปรแกรม pandas เพื่อโหลดข้อมูลและแปลงให้อยู่ในรูปแบบของ DataFrame โดยเก็บไว้ที่ตัวแปร df หากต้องการแสดงข้อมูลโรคเบาหวานสามารถทำได้โดยพิมพ์คำสั่ง head()

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

ข้อมูลโรคเบาหวานประกอบด้วย 8 Attribute และผลลัพธ์ที่ได้มีค่าเป็น 0 หรือ 1 ซึ่งอยู่ใน Attribute ที่ชื่อ Outcome ซึ่งก็คือ Output/Label/Class และหากต้องการที่จะทราบจำนวนของข้อมูลสามารถทำได้โดยใช้คำสั่ง shape

```
df.shape
```

```
(768, 9)
```

ข้อมูลโรคเบาหวานที่ใช้ในการทดลองประกอบด้วย 768 instance แต่ละ instance มีจำนวน 8 attribute และอีก 1 attribute เป็นผลลัพธ์

กำหนดข้อมูลที่จะนำไปใช้ในอัลกอริธึม KNN สามารถทำได้โดย

```
from sklearn.model_selection import train_test_split

     X = df.drop('Outcome',axis=1).values
     y = df['Outcome'].values
     X_train,X_test,y_train,y_test =
     train_test_split(X,y,test_size=0.4, \
                         random_state=42, stratify=y)
```

จากตัวอย่างข้อมูลที่เป็น Feature จะถูกเก็บไว้ที่ตัวแปร X และข้อมูลของ Class จะถูกเก็บไว้ที่ตัวแปร y จากนั้นใช้คำสั่ง train_test_split() เพื่อแบ่งข้อมูลเป็นชุดเรียนรู้ และชุดทดสอบ โดยกำหนดให้ขนาดของชุดทดสอบมีขนาดเป็น 40% โดยการกำหนด test_size=0.4

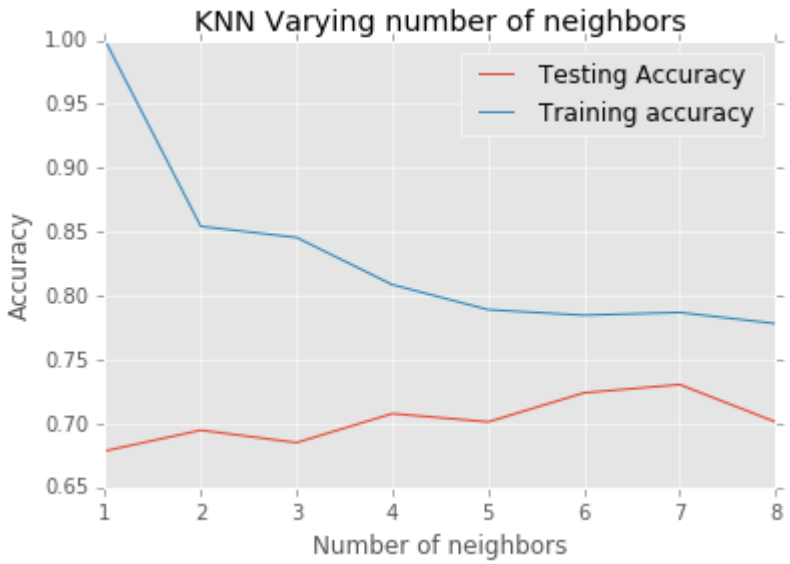
```
     from sklearn.neighbors import KNeighborsClassifier

     neighbors = np.arange(1,9)
     train_accuracy =np.empty(len(neighbors))
     test_accuracy = np.empty(len(neighbors))

     for i,k in enumerate(neighbors):
         #Setup a knn classifier with k neighbors
         knn = KNeighborsClassifier(n_neighbors=k)

         #Fit the model
         knn.fit(X_train, y_train)

        #Compute accuracy on the training set
        train_accuracy[i] = knn.score(X_train, y_train)

        #Compute accuracy on the test set
        test_accuracy[i] = knn.score(X_test, y_test)
```

จากตัวอย่างจะทำการทดสอบประสิทธิภาพของอัลกอริธึมโดยกำหนดจำนวนของ n_neighbors โดยใช้คำสั่ง neighbors = np.arange(1,9) ซึ่งผลลัพธ์ที่ได้จากคำสั่งนี้คือ

```
array([1, 2, 3, 4, 5, 6, 7, 8])
```

โปรแกรมจะทดสอบทั้งสิ้นจำนวน 8 รอบ และเก็บผลลัพธ์ไว้ที่ตัวแปร train_accuracy และ test_accuracy เพื่อใช้ในการแสดงผลลัพธ์

```
plt.title('KNN Varying number of neighbors')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```



ภาพประกอบที่ 15: กราฟแสดงประสิทธิภาพของอัลกอริธึม KNN เมื่อเปลี่ยนค่าพารามิเตอร์ n_neighbors

จากกราฟข้างต้น ประกอบด้วยข้อมูลจำนวน 2 เส้น เส้นสีแดง แทน Testing Accuracy และเส้นสีน้ำเงินแทน Training Accuracy จะเห็นความแตกต่างระหว่างการทดสอบกับ Training set และ Test set ดังนั้น ในการ Training ยิ่งกำหนดให้จำนวน n_neighbors มากเท่าไหร่ผลที่ได้จากการทดลองจะต่ำลงโดยกำหนดให้ n_neighbors=2 จะมีประสิทธิภาพสูงที่สุด ในทางกลับกันเมื่อทดสอบกับ Test set จำนวนของ n_neighbors ที่ให้ประสิทธิภาพสูงที่สุดมีค่าเท่ากับ 7 ดังนั้นการกำหนดค่าพารามิเตอร์ n_neighbors จึงขึ้นอยู่กับข้อมูลที่นำไปใช้

สร้างโมเดล KNN ด้วยค่า n_neighbor ที่ได้จากการทดลอง

จากการทดสอบทำให้ได้ค่า n_neibhbor=7 ซึ่งเป็นค่าที่ดีที่สุดในการทดสอบกับ Test set ดังนั้น หากต้องการทราบประสิทธิภาพของอัลกอริธึม KNN จึงต้องทดสอบโดยการสร้างโมเดลอีกครั้ง ดังตัวอย่างต่อไปนี้

```
knn = KneighborsClassifier(n_neighbors=7)
knn.fit(X_train,y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=1, n_neighbors=7, p=2,
 weights='uniform')
```

จากนั้นใช้คำสั่ง score() เพื่อแสดงค่าความถูกต้อง (Accuracy) ซึ่งผลลัพธ์ที่ได้คือ 73.05%

```
print("Accuracy", knn.score(X_test,y_test)*100)
```

```
('Accuracy', 73.05194805194806)
```

หากต้องการแสดงผลการทดลองด้วย Confusion Matrix สามารถทำได้ดังนี้

```
from sklearn.metrics import confusion_matrix

y_pred = knn.predict(X_test)
confusion_matrix(y_test,y_pred)
```

```
array([[165,  36],
       [ 47,  60]])
```

คำสั่ง Pandas Crosstab

ในโปรแกรม Pandas สามารถใช้คำสั่ง crosstab() เพื่อสร้าง Confusion Matrix ได้เช่นเดียวกัน สามารถทำได้ดังตัวอย่างต่อไปนี้

pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True)

Predicted	0	1	All
True			
0	165	36	201
1	47	60	107
All	212	96	308

แสดงผลลัพธ์จากการทดลองด้วย Classification Report

```
from sklearn.metrics import classification_report

print(classification_report(y_test,y_pred))
```

```
             precision    recall  f1-score   support

          0       0.78      0.82      0.80       201
          1       0.62      0.56      0.59       107

avg / total       0.73      0.73      0.73       308
```

การทดสอบค่าพารามิเตอร์ (Hyperparameter Tuning) ด้วยวิธี Grid Search

จากการทดสอบข้างต้นทำให้ได้ค่า n_neighbors=7 แต่การทดสอบสามารถทำได้หลายวิธี เช่นวิธี Grid Search เป็นต้น วิธีการทำ Grid Search สามารถทำได้ดังนี้

```
from sklearn.model_selection import GridSearchCV
import numpy as np

param_grid = {'n_neighbors':np.arange(1,50,2)}
```

ผลลัพธ์ที่ได้จากคำสั่ง np.arange(1,50,2) แสดงดังต่อไปนี้

```
array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35,
37, 39, 41, 43, 45, 47, 49])
```

```
knn = KNeighborsClassifier()
knn_cv= GridSearchCV(knn,param_grid,cv=5)
knn_cv.fit(X,y)
```

```
GridSearchCV(cv=5, error_score='raise',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
           metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'n_neighbors': array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19,
            21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
```

เมื่อกระบวนการของการทำ Grid Search เสร็จสิ้นสามารถใช้คำสั่งดังต่อไปนี้เพื่อแสดงผลลัพธ์

```
knn_cv.best_score_
```

```
0.7552083333333334
```

```
knn_cv.best_params_
```

```
{'n_neighbors': 13}
```

ผลลัพธ์ที่ได้คือเมื่อกำหนดให้พารามิเตอร์ n_neighbors=13 จะมีความถูกต้องที่ 75.52%

การจัดหมวดหมู่ชุดข้อมูล MNIST ด้วยอัลกอริธึม KNN

ตัวอย่างต่อไปนี้ เป็นการใช้อัลกอริธึม KNN เพื่อจัดหมวดหมู่ชุดข้อมูล MNIST โดยใช้โปรแกรม scipy ในการโหลดข้อมูล

```
from scipy.io import loadmat

mnist_raw = loadmat("mldata/mnist-original.mat")
mnist = {
    "data": mnist_raw["data"].T,
    "target": mnist_raw["label"][0],
    "COL_NAMES": ["label", "data"],
    "DESCR": "mldata.org dataset: mnist-original",
    }

X, y = mnist['data'], mnist['target']
```

จากตัวอย่างข้างต้นใช้คำสั่ง loadmat() ในการโหลดข้อมูลนามสกุล .mat และเก็บข้อมูลไว้ที่ตัวแปร X และ Label ไว้ที่ตัวแปร y ในกรณีนี้จะทำการสลับข้อมูล (shuffle) เพื่อให้ข้อมูลที่จะนำมา Visualization มีความหลากหลาย

```
import numpy as np

shuffle_index = np.random.permutation(70000)
X, y = X[shuffle_index], y[shuffle_index]
```

จากตัวอย่างทำการสลับข้อมูลทั้งหมด 70,000 ชุด ด้วยคำสั่ง permutation() โปรแกรมจะสลับและคืนค่าตำแหน่ง (index) เก็บไว้ในตัวแปร shuffle_index

```
X_train, X_test, y_train, y_test = X[:20000], X[69000:],
y[:20000], y[69000:]

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

```
((20000, 784), (1000, 784), (20000,), (1000,))
```

จากนั้น เลือกข้อมูลเพื่อใช้ในการเรียนรู้ (X_train) จำนวน 20,000 ชุด (x[:20000]) และเลือกข้อมูลเพื่อใช้ในการทดสอบ (X_test) จำนวน 1,000 ชุด (x[69000:]) โดยรูปภาพตัวเลขมีขนาด 28x28 จึงทำให้ Feature ที่ใช้มีทั้งสิ้น 784 attribute

ในความเป็นจริงข้อมูล mnist-origianl.mat จะกำหนดให้ข้อมูลชุดที่ 1-60000 ใช้สำหรับเรียนรู้ และชุดที่ 60001-70000 ใช้สำหรับทดสอบ

KNN Classifier

ในขั้นตอนนี้ ทำการเลือกใช้โมเดล KNN เพื่อทดสอบกับชุดข้อมูล MNIST โดยมีข้อมูลสำหรับเรียนรู้จำนวน 20,000 instance และ ข้อมูลสำหรับทดสอบจำนวน 1,000 instance การเลือกโมเดล KNN และกำหนดพารามิเตอร์ n_neighbors=1 ทำได้ดังตัวอย่างต่อไปนี้

```
from sklearn.neighbors import KNeighborsClassifier

knn = KneighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

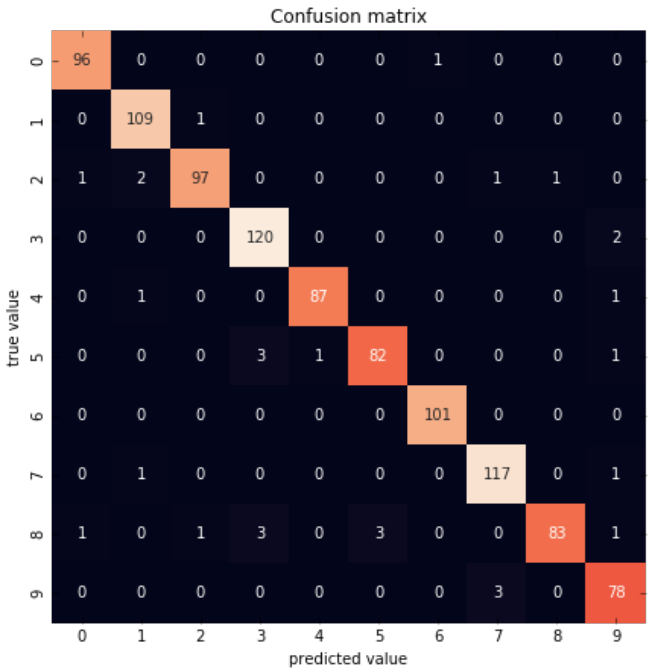
จากนั้นจึงสร้างโมเดลด้วยคำสั่ง fit() โมเดลที่สร้างจะถูกเก็บไว้ในตัวแปรที่ชื่อ knn

```
from sklearn.metrics import accuracy_score

y_model = knn.predict(X_test)
accuracy_score(y_test, y_model)
```

```
0.97
```

จากตัวอย่างข้างต้น นำโมเดล knn ไปพยากรณ์ข้อมูล X_test ด้วยคำสั่ง predict() และแสดงผลความถูกต้องด้วยคำสั่ง accuracy_score() และมีความถูกต้องที่ 0.97 หรือ 97% สามารถคำนวณค่า และแสดงตาราง Confusion Matrix ได้ดังต่อไปนี้

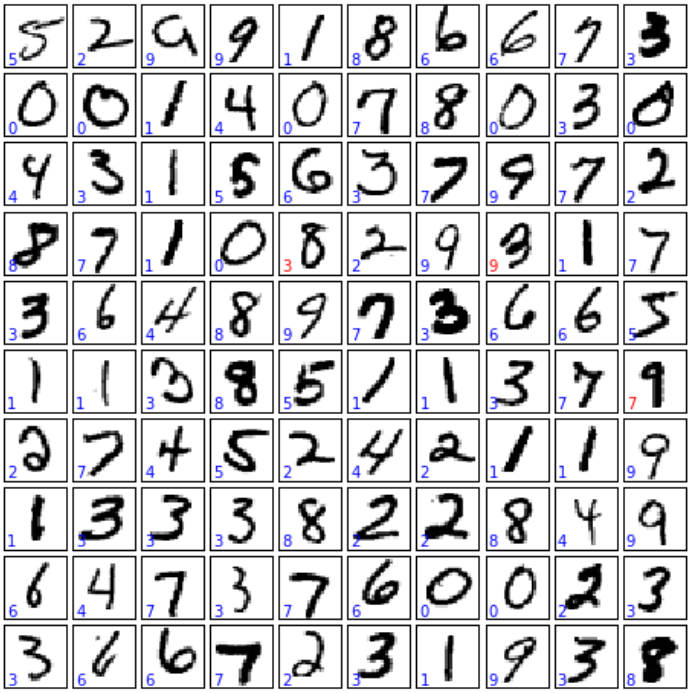
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

mat = confusion_matrix(y_test, y_model)

plt.figure(figsize=(7,7))
plt.title("Confusion matrix")
sns.heatmap(mat, square=True, annot=True, cbar=False, fmt="d")
plt.xlabel('predicted value')
plt.ylabel('true value')
plt.show()
```



จากตัวอย่าง เส้นทแยงคือผลลัพธ์ของโมเดล KNN ที่พยากรณ์ถูกต้อง และตัวเลขที่กระจายทั่ว Confusion Matrix คือตัวเลขที่พยากรณ์ผิดพลาด

สุดท้ายแล้วสามารถแสดงข้อมูลการพยากรณ์แบบ Visualization ได้ดังต่อไปนี้

```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(10, 10, figsize=(8, 8),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1, wspace=0.1))

for i, ax in enumerate(axes.flat):
     ax.imshow(X_test[i].reshape((28,28)), cmap='binary', \
                         interpolation='nearest')
     ax.text(0.05, 0.05, str(int(y_model[i])), \
                          transform=ax.transAxes, \
                          color='blue' if(y_test[i] == y_model[i])\
                          else 'red')

plt.show()
```



จากตัวอย่าง ตัวเลขสีน้ำเงินคือผลลัพธ์จากการพยากรณ์ที่ถูกต้อง และตัวเลขสีแดงคือการพยากรณ์ผิดพลาด

บทที่ 9
การจัดหมวดหมู่ข้อมูลด้วย Naive Bayes (Naive Bayes Classification)

Naive Bayes Classification เป็นการจัดหมวดหมู่ข้อมูลแบบที่มีการเรียนการสอน (Supervised Learning) โดยใช้หลักความน่าจะเป็นเข้ามาช่วยคำนวณ

ตัวอย่างต่อไปนี้ใช้ไลบรารี่ Seaborn ในการเรียกใช้ข้อมูล Iris Dataset

```
import seaborn as sns

iris = sns.load_dataset('iris')
print("Type", type(iris))

iris.head()
```

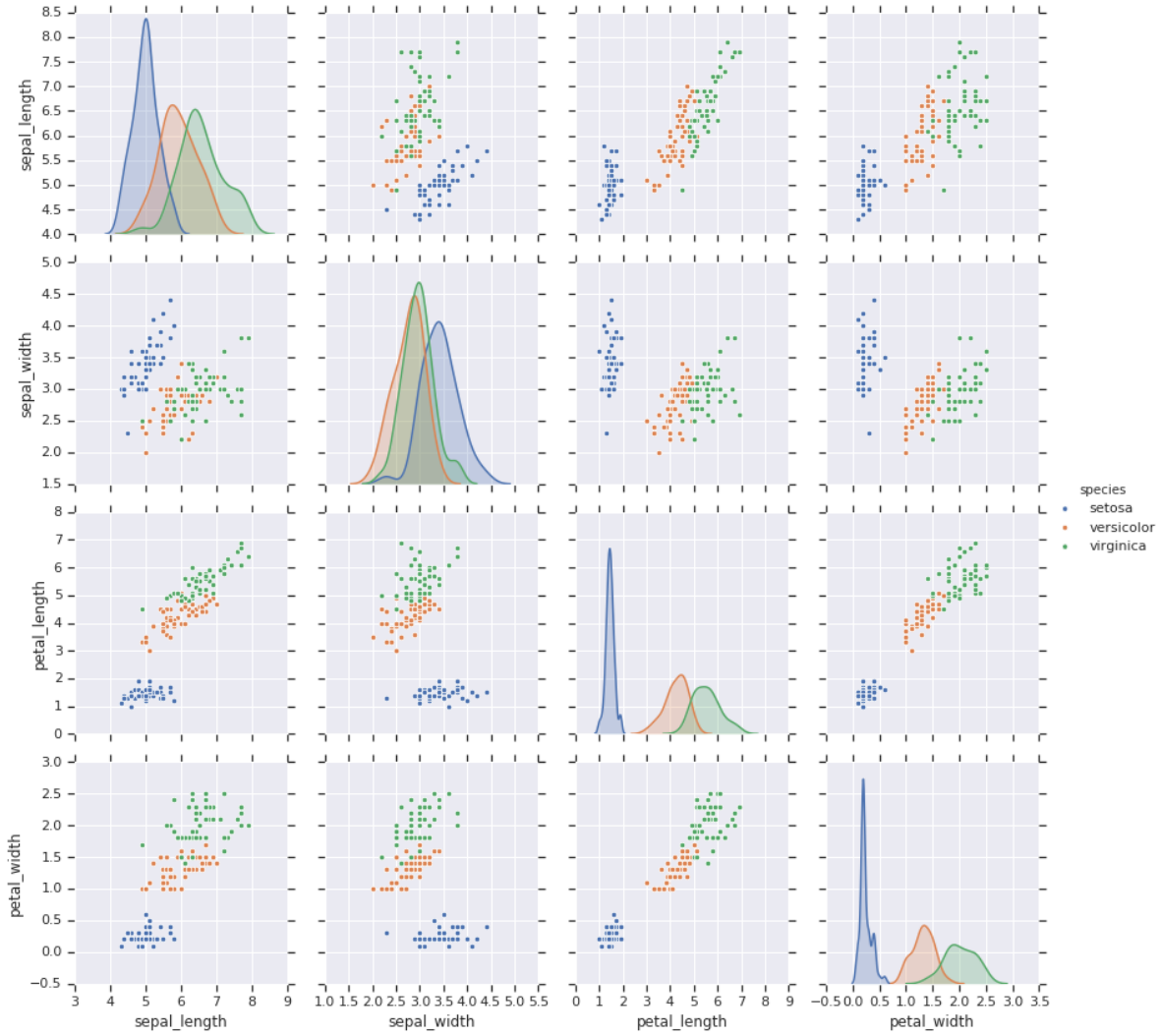
```
('Type', <class 'pandas.core.frame.DataFrame'>)
```

ข้อมูลที่โหลดมาจะจัดเก็บอยู่ในตัวแปร iris โดยที่ตัวแปร iris จะเก็บข้อมูลแบบ pandas DataFrame

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

สามารถใช้ไลบรารี่ Seaborn เพื่อ Visualization ข้อมูล Iris ได้ วิธีการพลอตทำได้ดังต่อไปนี้

```
import matplotlib.pyplot as plt

sns.set()
sns.pairplot(iris, hue='species',size=3)
plt.show()
```



การเตรียมข้อมูลเพื่อใช้ในการเรียนรู้

ชุดข้อมูล Iris มีจำนวน 150 instance และแบ่งออกเป็น 3 กลุ่ม ในตัวอย่างต่อไปนี้จะใช้คำสั่ง train_test_split เพื่อแบ่งข้อมูลออกเป็นชุด Train และ Test

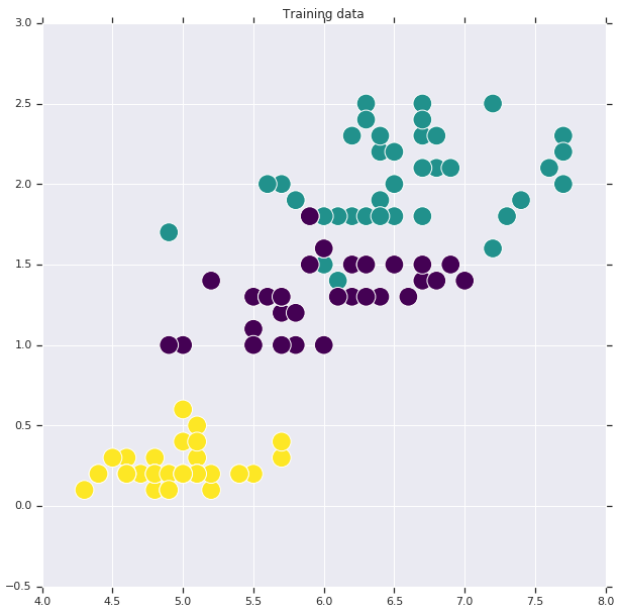
```
from sklearn.cross_validation import train_test_split

Xtrain, Xtest, ytrain, ytest = train_test_split(X_iris, \
                                                y_iris, random_state=1)
```

จากนั้นสามารถพลอตข้อมูล iris เพื่อดูการกระจายของข้อมูล แต่ทั้งนี้จะต้องแปลง Label ที่อยู่ในรูปแบบของตัวอักษร ให้เป็นตัวเลขเสียก่อน

```
ytrain[ytrain.iloc[0:] == 'versicolor'] = 1
ytrain[ytrain.iloc[0:] == 'virginica'] = 2
ytrain[ytrain.iloc[0:] == 'setosa'] = 3
```

เมื่อแปลง Label ให้มีค่าเป็น 1,2 และ 3 จากนั้นจึงพลอตข้อมูล iris โดยใช้คำสั่งดังต่อไปนี้

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,10))
plt.scatter(Xtrain.iloc[:,0:1], Xtrain.iloc[:,3:4], \
                c=ytrain[:], s=350, cmap='viridis')
plt.title('Training data')
plt.show()
```



สร้างโมเดล Naive Bayes

สามารถใช้คำสั่ง GaussianNB() ซึ่งก็คือ Gaussian Naive Bayes ในการสร้างโมเดล และคำสั่ง fit() สำหรับเรียนรู้ข้อมูล

```
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()
model.fit(Xtrain, ytrain)
```

```
GaussianNB(priors=None)
```

พยากรณ์ข้อมูลด้วยโมเดล Naive Bayes และแสดงประสิทธิภาพของโมเดล

การพยากรณ์ทำได้โดยใช้คำสั่ง predict() และการแสดงผลความถูกต้องใช้คำสั่ง accuracy_score() สุดท้ายแล้วใช้ Confusion Matrix เพื่อดูลักษณะการพยากรณ์ข้อมูล ดังตัวอย่างต่อไปนี้

```
from sklearn.metrics import accuracy_score

y_model = model.predict(Xtest)
accuracy_score(ytest, y_model)
```

```
0.9736842105263158
```

ผลลัพธ์ที่ได้จากการพยากรณ์คือ 97.37% จากนั้นใช้คำสั่งดังต่อไปนี้เพื่อแสดงค่า Confusion Matrix

```
from sklearn.metrics import confusion_matrix

mat = confusion_matrix(ytest, y_model)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', \
               cbar=False, cmap='viridis',
               xticklabels=['setosa', 'versicolor', 'virginica'],\
               yticklabels=['setosa', 'versicolor', 'virginica'])\


plt.title('Confusion matrix')
plt.xlabel('true label')
plt.ylabel('predicted label');
plt.show()
```

Confusion matrix
predicted label
setosa
versicolor
virginica
13
0
0
0
15
0
0
1
9
setosa
versicolor
virginica
true label

บทที่ 10
การวิเคราะห์องค์ประกอบหลัก
(Principal Component Analysis)

การวิเคราะห์องค์ประกอบหลัก (Principal Component Analysis: PCA) เป็นวิธีที่ใช้เพื่อวิเคราะห์ข้อมูลที่มีหลายตัวแปร (Variable) เพื่อหาความสัมพันธ์ของตัวแปรเหล่านั้น ทำให้เกิดการลดขนาดเมตริกซ์ (Matrix) ที่มีความซับซ้อนเล็กลงง่ายต่อการอธิบาย[1] นักวิจัยจึงใช้วิธี PCA เพื่อนำมาลดขนาดของคุณลักษณะพิเศษ (Feature) ให้มีเล็กลง [2] ทำให้ลดเวลาในการสร้างโมเดล

การสร้างโมเดล PCA

ตัวอย่างต่อไปนี้ใช้ชุดข้อมูล Iris ในการทดสอบ ซึ่งชุดข้อมูล Iris มีทั้งสิ้น 4 attribute ซึ่งโหลดข้อมูลโดยใช้ไลบรารี่ Seaborn

```
import seaborn as sns

iris = sns.load_dataset('iris')
X_iris = iris.drop('species', axis=1)
y_iris = iris['species']
```

ข้อมูล feature ทั้ง 4 column จะถูกจัดเก็บไว้ที่ตัวแปร X_iris และข้อมูล Label จะถูกเก็บอยู่ที่ตัวแปร y_iris

```
from sklearn.decomposition import PCA

model = PCA(n_components=2)
model.fit(X_iris)
```

```
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
      svd_solver='auto', tol=0.0, whiten=False)
```

1 การวิเคราะห์ส่วนประกอบสำคัญ: http://www.edu.tsu.ac.th/major/eva/files/journal/PRINCIPAL.pdf
2 PCA: https://www.gotoknow.org/posts/566063

การสร้างโมเดลจะเรียกใช้คำสั่ง PCA() เพื่อกำหนดค่าพารามิเตอร์ และคำสั่ง fit() เพื่อเรียนรู้ลักษณะของข้อมูล โดยโมเดลจะถูกเก็บไว้ที่ตัวแปรชื่อ model

สามารถตรวจสอบจำนวนของ component ที่ได้จากการคำนวณด้วย PCA โดยใช้คำสั่งดังต่อไปนี้

```
print("check number of components", model.n_components_)
```

```
('check number of components', 2)
```

ขั้นตอนต่อไปคือเปลี่ยนรูปแบบของข้อมูล (Transform the data) ให้ตรงกับรูปแบบของ scikit-learn จากตัวอย่างใช้วิธีการดังต่อไปนี้

```
X_2D = model.transform(X_iris)
print("show first row of data", X_2D[0,:])

print model.transform(X_iris.iloc[0,:].
          as_matrix().reshape(1, -1))
```

จากนั้นทดสอบข้อมูล X_2D โดยเลือกแถวที่ 0 มาแสดง ดังตัวอย่างต่อปนี้

```
print("show first row of data", X_2D[0,:])
('show first row of data', array([-2.68412563,  0.31939725]))
```

ตัวอย่างต่อไปแสดงให้เห็นถึงการแปลงข้อมูล X_iris เฉพาะแถวที่ 1 โดยใช้คำสั่ง transform()

```
print model.transform(X_iris.iloc[0,:].as_matrix().reshape(1, -1))
```

```
[[-2.68412563  0.31939725]]
```

การเพิ่มข้อมูลจากตัวแปรเข้าไปเก็บเพิ่มใน DataFrame

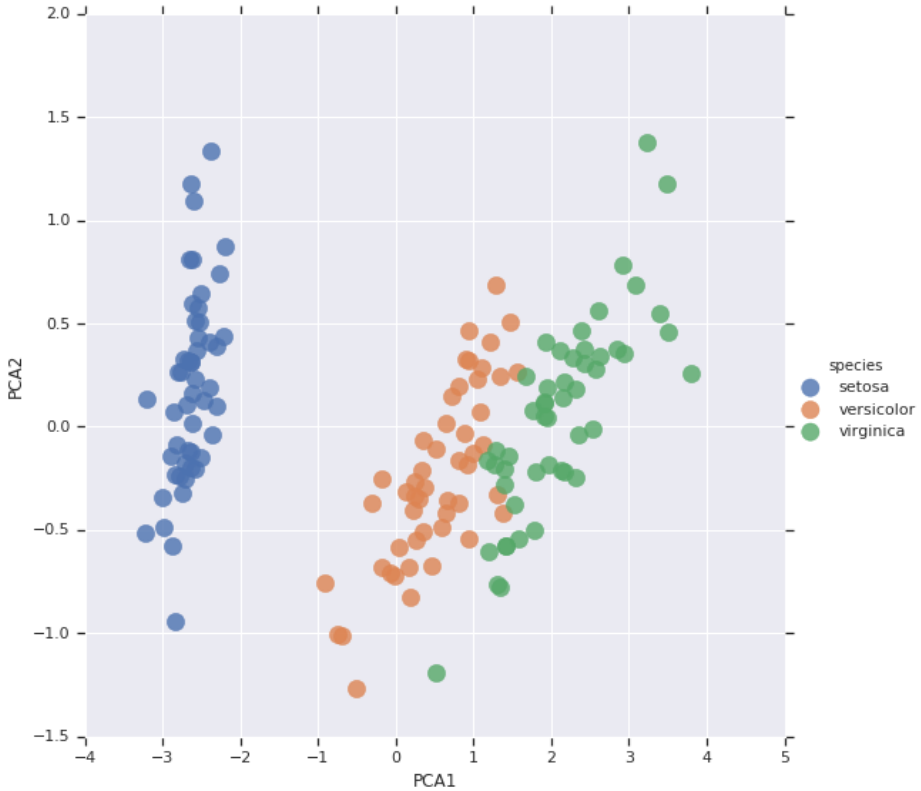
จากตัวแปร iris ที่โหลดมาโดยใช้ไลบรารี่ seaborn สามารถเพิ่มข้อมูลเข้าไปใหม่ได้ ดังตัวอย่างต่อไปนี้

```
iris['PCA1'] = X_2D[:,0]
iris['PCA2'] = X_2D[:,1]
```

จากตัวอย่างข้างต้น ได้เพิ่มข้อมูล X_2D เข้าไปในตัวแปร iris จำนวน 2 คอลัมน์ โดยกำหนดให้ชื่อของข้อมูลคือ PCA1 และ PCA2

การพลอตข้อมูลด้วยไลบรารี่ seaborn เพื่อดูลักษณะการกระจายของข้อมูลที่ได้จากการคำนวณด้วยวิธี PCA

```
import seaborn as sns

sns.lmplot("PCA1", "PCA2", hue='species', data=iris, \
 fit_reg=False, size=8, scatter_kws={"s": 150})
```



เปลี่ยนจำนวนของ Components

สามารถเปลี่ยนจำนวนของ Components ด้วยการกำหนดที่ตัวแปร n_components ดังตัวอย่างต่อไปนี้

```
from sklearn.decomposition import PCA

model = PCA(n_components=5)
model.fit(X_iris)
X_2D = model.transform(X_iris)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-58-acad37c053c8> in <module>()
      2
      3 model = PCA(n_components=5)    # 2. Instantiate the model with
hyperparameters
```

```
----> 4 model.fit(X_iris)              # 3. Fit to data. Notice y is not
specified!
      5 X_2D = model.transform(X_iris) # 4. Transform the data to two
dimenstions

/usr/local/lib/python2.7/dist-packages/sklearn/decomposition/pca.pyc in
fit(self, X, y)
    305             Returns the instance itself.
    306         """
--> 307         self._fit(X)
    308         return self
    309

/usr/local/lib/python2.7/dist-packages/sklearn/decomposition/pca.pyc in
_fit(self, X)
    366         # Call different fits for either full or truncated SVD
    367         if svd_solver == 'full':
--> 368             return self._fit_full(X, n_components)
    369         elif svd_solver in ['arpack', 'randomized']:
    370             return self._fit_truncated(X, n_components, svd_solver)

/usr/local/lib/python2.7/dist-packages/sklearn/decomposition/pca.pyc in
_fit_full(self, X, n_components)
    381             raise ValueError("n_components=%r must be between 0 and "
    382                              "n_features=%r with svd_solver='full'"
--> 383                              % (n_components, n_features))
    384
    385         # Center data

ValueError: n_components=5 must be between 0 and n_features=4 with
svd_solver='full'
```

ในกรณีที่กำหนดให้ n_components มีค่าเท่ากับ 5 จะเกิด error ดังข้อความ error ข้างต้น ทั้งนี้เนื่องจากชุดข้อมูล Iris มีเพียง 4 attribute เท่านั้น ดังนั้น ในการคำนวณด้วยวิธี PCA จะต้องคำนึงถึงจำนวนของ attribute เป็นหลัก ซึ่งไม่สามารถกำหนดให้ n_components มีค่ามากกว่าจำนวนของ attribute

```
from sklearn.decomposition import PCA


model = PCA(n_components=3)
model.fit(X_iris)
X_2D = model.transform(X_iris)

iris['PCA1'] = X_2D[:,0]
iris['PCA2'] = X_2D[:,1]
iris['PCA3'] = X_2D[:,2]

iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species	PCA1	PCA2	PCA3
0	5.1	3.5	1.4	0.2	setosa	-2.684126	0.319397	-0.027915
1	4.9	3.0	1.4	0.2	setosa	-2.714142	-0.177001	-0.210464
2	4.7	3.2	1.3	0.2	setosa	-2.888991	-0.144949	0.017900
3	4.6	3.1	1.5	0.2	setosa	-2.745343	-0.318299	0.031559
4	5.0	3.6	1.4	0.2	setosa	-2.728717	0.326755	0.090079

จากตัวอย่างข้างต้น ทดสอบด้วยการกำหนด n_components ให้มีค่าเท่ากับ 3 และแสดงข้อมูลที่ได้จากการคำนวณด้วยวิธี PCA ซึ่งเก็บอยู่ในตัวแปร PCA1, PCA2 และ PCA3

สร้างโมเดล Naive Bayes ด้วยคุณลักษณะพิเศษที่ได้จาก PCA

เพื่อทดสอบประสิทธิภาพของอัลกอริธึม ในกรณีนี้ได้นำคุณลักษณะพิเศษที่ได้จากการคำนวณด้วยวิธี PCA มาใช้เพื่อสร้างโมเดลด้วยวิธี Naive Bayes

```
from sklearn.cross_validation import train_test_split

X_iris = iris.drop('species', axis=1)
y_iris = iris['species']

Xtrain, Xtest, ytrain, ytest = train_test_split(X_iris,\
                                                y_iris, random_state=1)
```

เริ่มต้นด้วยการแบ่งข้อมูลออกเป็นชุดเรียนรู้ และชุดทดสอบ ด้วยคำสั่ง train_test_split() จากนั้นใช้คำสั่ง Xtrain.head() เพื่อดูข้อมูลและลำดับของข้อมูลที่ผ่านการสลับจากคำสั่ง train_test_split()

```
X_train.head()
```

	sepal_length	sepal_width	petal_length	petal_width	PCA1	PCA2	PCA3
54	6.5	2.8	4.6	1.5	1.088103	0.074591	-0.307758
108	6.7	2.5	5.8	1.8	2.321229	-0.243832	-0.348304
112	6.8	3.0	5.5	2.1	2.165592	0.216276	0.033327
17	5.1	3.5	1.4	0.3	-2.648297	0.311849	0.026668
119	6.0	2.2	5.0	1.5	1.300792	-0.761150	-0.344995

ขั้นตอนถัดไป เลือกข้อมูลในคอลัมน์ PCA1, PCA2, PCA3 เพื่อใช้สำหรับการเรียนรู้ และการทดสอบ

```
Xtrain = Xtrain.ix[:,['PCA1','PCA2','PCA3']]
Xtrain.head()
```

	PCA1	PCA2	PCA3
54	1.088103	0.074591	-0.307758
108	2.321229	-0.243832	-0.348304
112	2.165592	0.216276	0.033327
17	-2.648297	0.311849	0.026668
119	1.300792	-0.761150	-0.344995

```
Xtest = Xtest.ix[:,['PCA1','PCA2','PCA3']]
Xtest.head()
```

	PCA1	PCA2	PCA3
14	-2.644750	1.178765	-0.151628
98	-0.906470	-0.756093	-0.012600
75	0.900174	0.328504	-0.316209
16	-2.623528	0.810680	0.138183
131	3.230674	1.374165	-0.114548

จากนั้นนำข้อมูลจากตัวแปร Xtrain และ Xtest ข้างต้น มาสร้างโมเดลด้วยวิธี Naive Bayes ดังตัวอย่างต่อไปนี้

```
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()
model.fit(Xtrain, ytrain)
```

```
GaussianNB(priors=None)
```

การพยากรณ์คุณลักษณะพิเศษที่ได้จาก PCA ด้วยอัลกอริธึม Naive Bayes และประสิทธิภาพจากการพยากรณ์

ทำการพยากรณ์ข้อมูลด้วยคำสั่งดังต่อไปนี้

```
y_model = model.predict(Xtest)
```

จากนั้นทำการคำนวณหาค่าความถูกต้อง ดังตัวอย่างต่อไปนี้

```
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model)
```

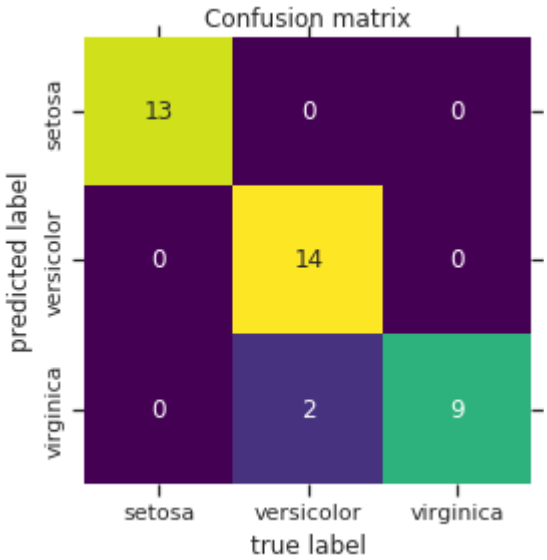
```
0.9473684210526315
```

ความถูกต้องที่ได้จากการนำคุณลักษณะพิเศษของ PCA จำนวน 3 components ไปทำการสร้างโมเดล และทดสอบ ปรากฏว่ามีความถูกต้อง 94.74%

```
from sklearn.metrics import confusion_matrix

mat = confusion_matrix(ytest, y_model)

sns.heatmap(mat.T, square=True, annot=True, fmt='d',
          cbar=False, cmap='viridis',
          xticklabels=['setosa', 'versicolor', 'virginica'],
yticklabels=['setosa', 'versicolor', 'virginica'])
```

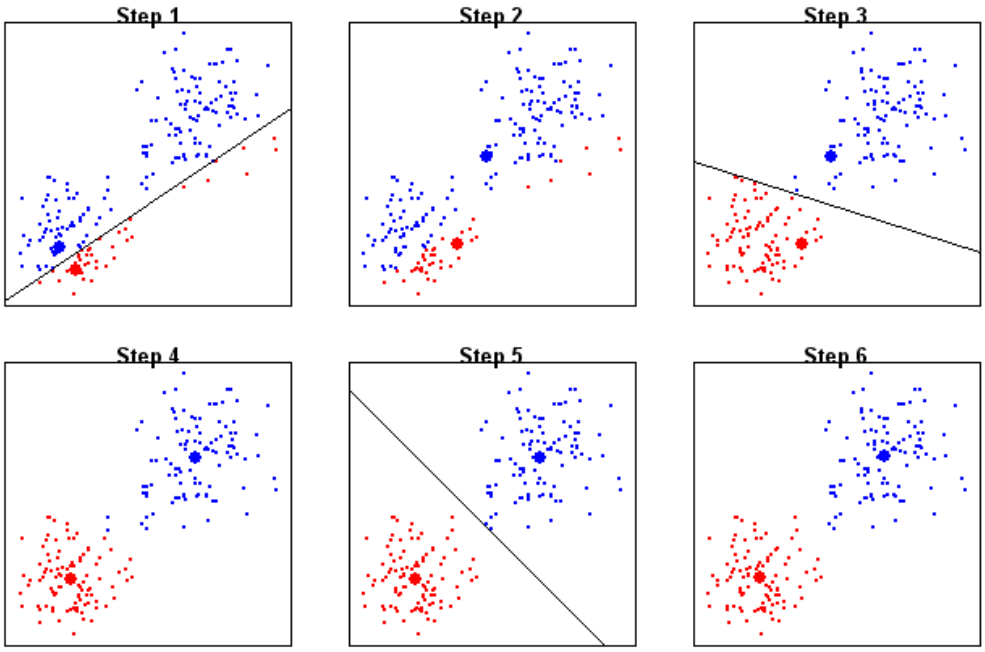
สุดท้ายแสดงผลการคำนวณค่า Confusion matrix เพื่อดูความถูกต้องของการพยากรณ์ข้อมูล

```
plt.title('Confusion matrix')
plt.xlabel('true label')
plt.ylabel('predicted label');
plt.show()
```



บทที่ 11
การจัดกลุ่มด้วยอัลกอริธึม K-Means (K-Means Clustering)

อัลกอริธึม K-Means เป็นวิธีที่ใช้สำหรับค้นหาจำนวนของคลัสเตอร์ (Cluster) จากข้อมูลที่ไม่ปรากฏคลาส (Class) หรือไม่มี Label ซึ่งจะเรียกว่า Unlabeled Data วิธีนี้จึงเป็นวิธี Unsupervised Learning หรือวิธี Clustering



ภาพประกอบที่ 16: ตัวอย่างการจัดกลุ่มด้วยวิธี K-Means โดยแสดงให้ดูทีละขั้นตอน

จากภาพประกอบที่ 16 หากดูด้วยตาเปล่าข้อมูลสามารถแบ่งออกเป็น 2 กลุ่ม โดยจำลองให้เป็นกลุ่มสีแดง และกลุ่มสีน้ำเงิน

Step 1 ต้องสุ่มเลือกค่ากลาง (Centroid) เพื่อใช้เป็นตัวแทนของกลุ่มสีแดง และสีน้ำเงิน

เพื่อใช้สำหรับคำนวณหาค่าระยะห่าง (Distance Measurement) เพื่อเปรียบเทียบระหว่างจุดข้อมูลและค่า Centroid ทั้ง 2 กลุ่ม (สีแดง และสีน้ำเงิน) หากจุดนั้นใกล้กลุ่มใดที่สุดจะถูกกำหนด (Assign) ให้อยู่ในกลุ่มนั้น เช่น หากใกล้กลุ่มสีแดง จุดนั้นก็จะถูกกำหนดให้เป็นสีแดง

Step 2 เมื่อเปลี่ยนค่าให้กับทุกจุดข้อมูลเป็นที่เรียบร้อย จากนั้นให้คำนวณเพื่อหาค่า Centroid ใหม่ ในกรณีที่คำนวณหาค่า Centroid ของกลุ่มสีน้ำเงิน จะนำข้อมูลทุกจุดที่เป็นสีน้ำเงินมาคำนวณ เช่นเดียวกันกับสีแดง ดังนั้น จะได้จุด Centroid ใหม่ ที่ใช้เป็นตัวแทนของกลุ่มต่อไป

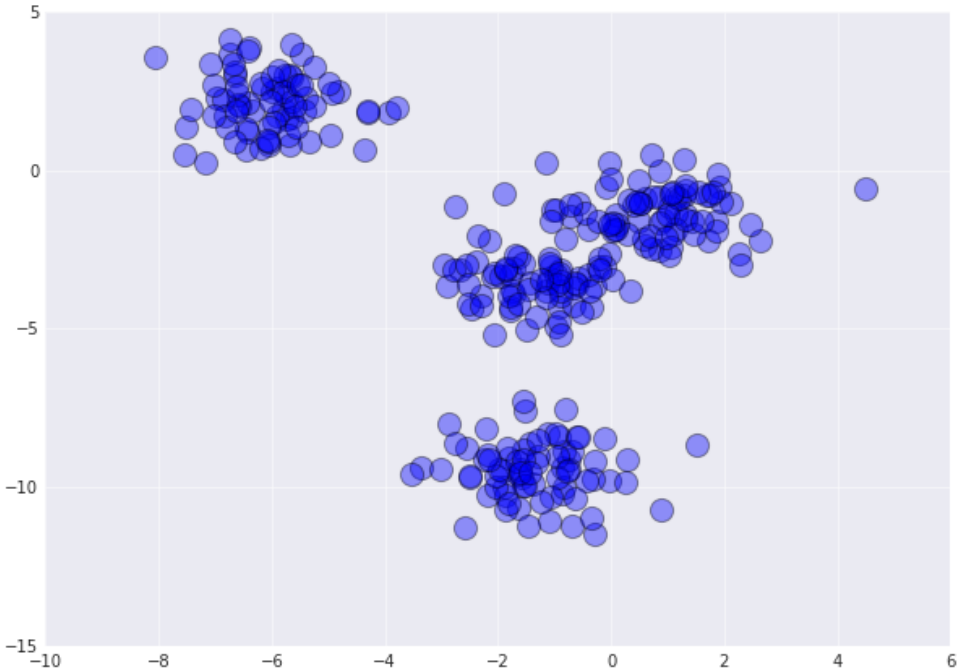
Step 3 คำนวณหาค่า Distance ระหว่าง Centroid ทั้ง 2 กลุ่ม และจุดข้อมูลทุกจุด เพื่อจัดกลุ่มข้อมูลใหม่ ทำไปจนกระทั่งจุดแต่ละจุดไม่มีการเปลี่ยนแปลง หรือมีการเปลี่ยนแปลงน้อยที่สุด (Step 4-5) โปรแกรมจะหยุดการทำงาน

Step 6 เมื่อไม่มีการเปลี่ยนแปลงกลุ่มของข้อมูล ให้คำนวณหา Centroid เพื่อใช้เป็นโมเดล สามารถนำไปทดสอบกับชุดข้อมูล Test set และหากได้ผลดี ก็สามารถนำไปใช้งานจริงได้

จำลองข้อมูลเพื่อใช้ในอัลกอริธึม K-Means

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets.samples_generator import make_blobs

rng = np.random.RandomState(0)
colors = rng.rand(300)

X, y_true = make_blobs(n_samples=300, centers=4,
                       cluster_std=0.85, random_state=2)

plt.style.use('seaborn-darkgrid')
plt.figure(figsize=(10,7))
plt.scatter(X[:, 0], X[:, 1], s=200, alpha=0.4,
            cmap='viridis');
plt.show()
```

จากตัวอย่าง ทำการจำลองข้อมูลด้วยคำสั่ง make_blobs() โดยกำหนดให้มีข้อมูลทั้งสิ้น 300 ชุดข้อมูล (n_samples) และแบ่งข้อมูลออกเป็น 4 กลุ่ม (centers) จากนั้นทำการ Visualization เพื่อดูลักษณะการกระจายของข้อมูล ดังรูปภาพต่อไปนี้



สร้างโมเดลของอัลกอริธึม K-Means

```
from sklearn.cluster import Kmeans

kmeans = Kmeans(n_clusters=4)
kmeans.fit(X)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=4, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

การพยากรณ์ด้วยอัลกอริธึม K-Means

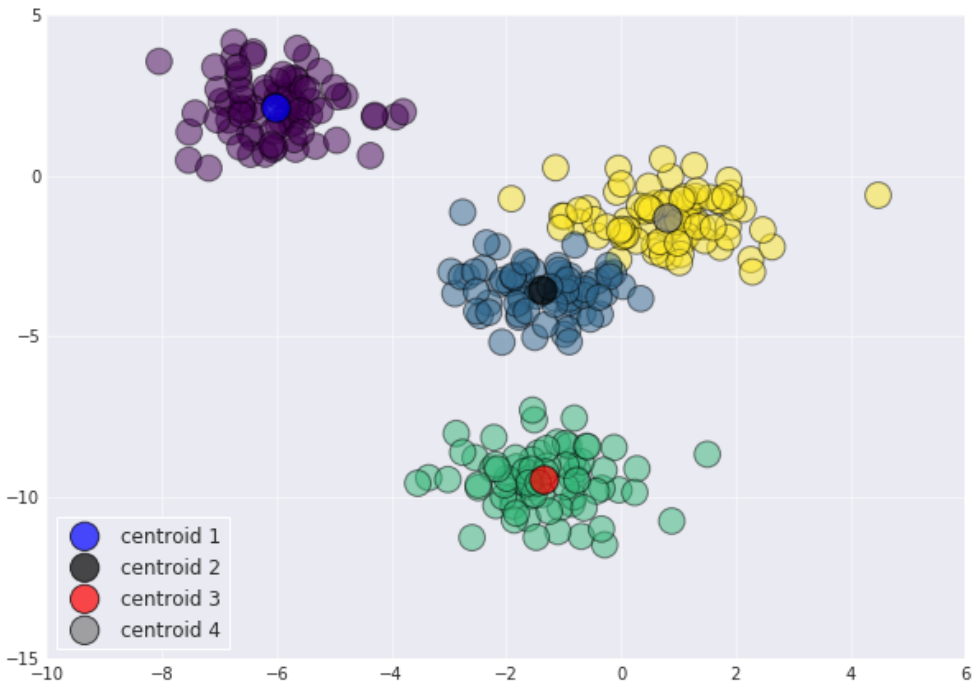
สำหรับการพยากรณ์ด้วย K-Means ในโปรแกรม scikit-learn จะใช้ฟังก์ชัน predict() เช่นเดียวกับอัลกอริธึมอื่น ๆ

```
y_kmeans = kmeans.predict(X)
print("output", y_kmeans[0:20])
```

```
('output', array([0, 0, 1, 0, 3, 2, 0, 2, 1, 0, 3, 0, 0, 2, 2, 3, 1, 2, 3, 0],
    dtype=int32))
```

จากนั้นแสดงผลการพยากรณ์จำนวน 20 ผลลัพธ์ เพื่อตรวจสอบคำตอบ จากนั้นทำการพลอตกราฟ เพื่อดูจุด Centroid และการแบ่งกลุ่มข้อมูล

```
plt.figure(figsize=(10,7))

# plot group of data
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=250, alpha=0.5,
            cmap='viridis')
centers = kmeans.cluster_centers_

# plot cluster/centroid
plt.scatter(centers[0, 0], centers[0, 1], c='blue', s=300, alpha=0.7,
      label='centroid 1')
plt.scatter(centers[1, 0], centers[1, 1], c='black', s=300, alpha=0.7,
      label='centroid 2')
plt.scatter(centers[2, 0], centers[2, 1], c='red', s=300, alpha=0.7,
      label='centroid 3')
plt.scatter(centers[3, 0], centers[3, 1], c='gray', s=300, alpha=0.7,
      label='centroid 4')

plt.legend(frameon=True, loc='lower left')
plt.show()
```



สร้างข้อมูลใหม่เพื่อทดสอบการแบ่งกลุ่มด้วยอัลกอริธึม K-Means

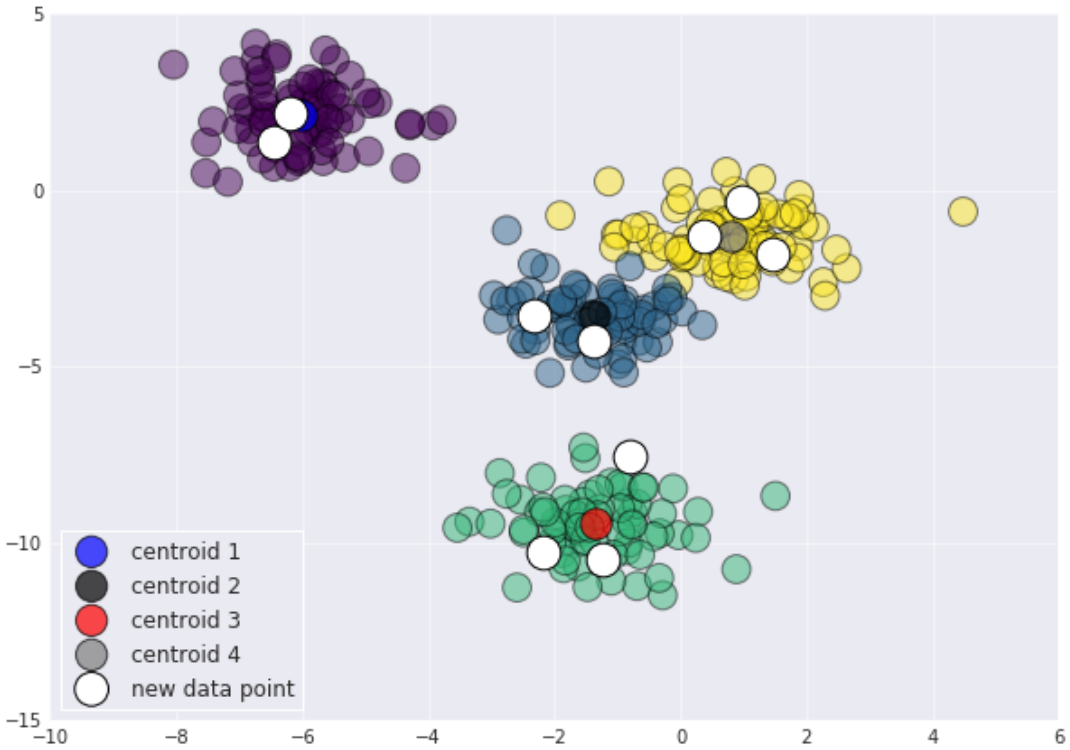
ทำการสร้างข้อมูลจำนวน 10 ชุดเพื่อใช้สำหรับการทดสอบโมเดลของ K-Means ดังตัวอย่างต่อไปนี้

```
X_test, y_test_true = make_blobs(n_samples=10, centers=4,
                                 cluster_std=0.85, random_state=2)

X_test.shape
```

```
(10, 2)
```

ขั้นตอนถัดไป ทำการพลอตข้อมูลที่สร้างขึ้นมาใหม่ เพื่อให้ทราบว่าข้อมูลที่สร้างขึ้นมาไปกระจายอยู่จุดใดของกราฟ สามารถทำได้ดังต่อไปนี้

```
plt.figure(figsize=(10,7))

# plot group of data
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=250, alpha=0.5,
                  cmap='viridis')
centers = kmeans.cluster_centers_

# plot cluster/centroid
plt.scatter(centers[0, 0], centers[0, 1], c='blue', s=300, alpha=0.7,
      label='centroid 1')
plt.scatter(centers[1, 0], centers[1, 1], c='black', s=300, alpha=0.7,
      label='centroid 2')
plt.scatter(centers[2, 0], centers[2, 1], c='red', s=300, alpha=0.7,
      label='centroid 3')
plt.scatter(centers[3, 0], centers[3, 1], c='gray', s=300, alpha=0.7,
      label='centroid 4')

# plot new data point
plt.scatter(X_test[:, 0], X_test[:, 1], c='white', s=350, alpha=1,
      label='new data point')
plt.legend(frameon=True, loc='lower left')

plt.show()
```



พยากรณ์ข้อมูลที่สร้างขึ้นมาใหม่ด้วยอัลกอริธึม K-Means

ในขั้นตอนการพยากรณ์จะทำการพยากรณ์ข้อมูล X_test ที่มีจำนวนทั้งสิ้น 10 ชุดข้อมูล โดยผลลัพธ์ที่ได้จากการพยากรณ์จะเก็บไว้ที่ตัวแปร y_test_kmeans

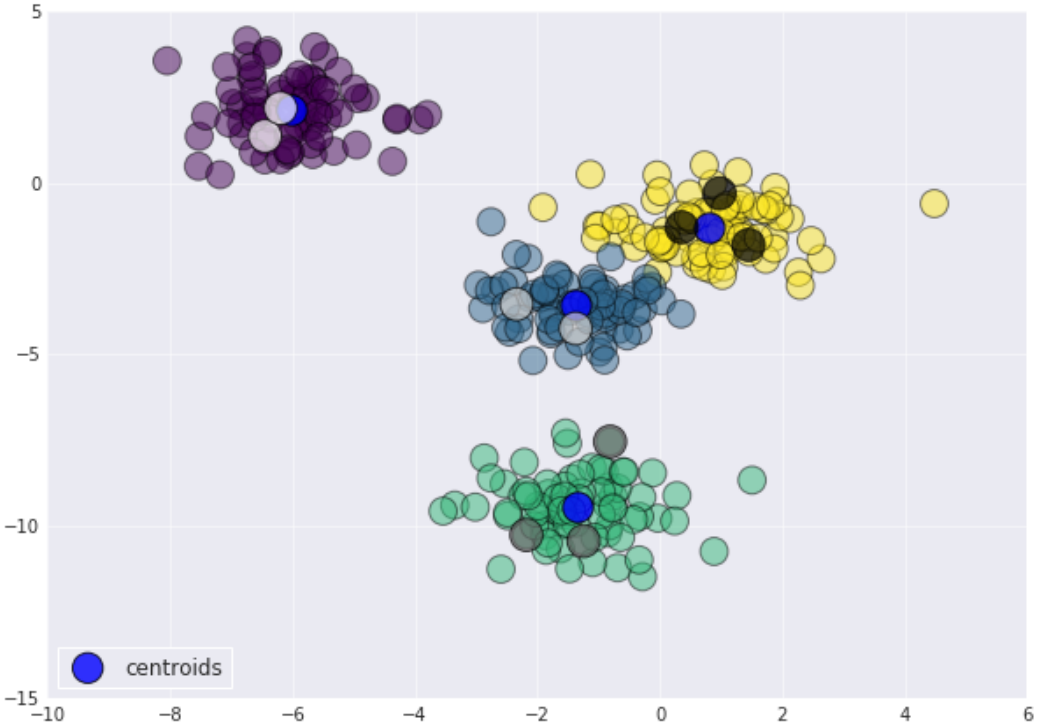
```
y_test_kmeans = kmeans.predict(X_test)
y_test_kmeans
```

```
array([1, 3, 1, 3, 2, 2, 0, 3, 2, 0], dtype=int32)
```

จากนั้นพลอตกราฟเพื่อแสดงผลแบบ Visualization

```
plt.figure(figsize=(10,7))

# plot group of data
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=250, alpha=0.5,
                cmap='viridis')
centers = kmeans.cluster_centers_

# plot cluster/centroid
plt.scatter(centers[:, 0], centers[:, 1], c='blue', s=300,
                alpha=0.8, label='centroids')
```

```
# plot new data point
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test_kmeans,
                s=350, alpha=0.7)
plt.legend(frameon=True, loc='lower left')

plt.show()
```



บทที่ 12
การรู้จำใบหน้า
(Face Recognition)

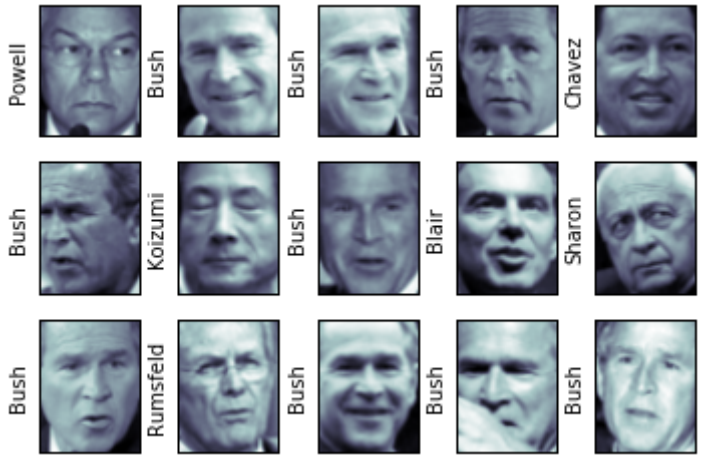
สำหรับการรู้จำใบหน้า ในกรณีนี้ได้ทดสอบกับชุดข้อมูล lfw โดยใช้วิธี Principal Component Analysis (PCA) เพื่อลดขนาดของ Feature และส่งต่อไปยัง Support Vector Machine (SVM) เพื่อเรียนรู้และสร้างโมเดล การรู้จำใบหน้าสามารถทำได้ดังต่อไปนี้

```
from sklearn.datasets import fetch_lfw_people

faces = fetch_lfw_people(min_faces_per_person=60)
print(faces.target_names)
print(faces.images.shape)
```

```
['Ariel Sharon' 'Colin Powell' 'Donald Rumsfeld' 'George W Bush'
 'Gerhard Schroeder' 'Hugo Chavez' 'Junichiro Koizumi' 'Tony Blair']
(1348, 62, 47)
```

ข้อมูลใบหน้าที่นำมาใช้ในการรู้จำ (Recognition) สามารถโหลดได้จาก scikit-learn โดยใช้คำสั่ง fetch_lfw_people จากนั้นทำการพลอตเพื่อดูรูปภาพบุคคล และรายชื่อของแต่ละบุคคล ที่นำมาใช้ในการรู้จำใบหน้า

```
import matplotlib.pyplot as plt

#plt.figure(figsize=(15,15))
fig, ax = plt.subplots(3, 5)
for i, axi in enumerate(ax.flat):
    axi.imshow(faces.images[i], cmap='bone')
    axi.set(xticks=[], yticks=[])
    axi.set_ylabel(faces.target_names[faces.target[i]].split()
                   [-1], color='black')
plt.show()
```



ภาพประกอบที่ 17: ตัวอย่างใบหน้าจากชุดข้อมูล lfw ที่นำมาใช้ในการรู้จำใบหน้า

สร้างโมเดลของอัลกอริธึม SVM

ในกรณีนี้ การสร้างโมเดลด้วย SVM จะใช้ข้อมูลที่ได้จากการคำนวณด้วย PCA โดยกำหนดให้ n_components=150 ดังนั้น ข้อมูลทั้งหมด 150 component จึงเรียกว่าเป็นคุณลักษณะพิเศษ (Feature Extraction) ที่จะส่งต่อไปยัง SVM เพื่อเรียนรู้และสร้างออกมาเป็นโมเดล ดังตัวอย่างต่อไปนี้

```
from sklearn.svm import SVC
from sklearn.decomposition import RandomizedPCA
from sklearn.pipeline import make_pipeline

pca = RandomizedPCA(n_components=150, whiten=True,
     random_state=42)

svc = SVC(kernel='rbf', class_weight='balanced')
model = make_pipeline(pca, svc)
```

จากนั้น แบ่งข้อมูลออกเป็น 2 ส่วน เพื่อใช้สำหรับการเรียนรู้ และทดสอบโดยใช้คำสั่ง train_test_split()

```
from sklearn.cross_validation import train_test_split

Xtrain, Xtest, ytrain, ytest = train_test_split(faces.data,
                                faces.target, random_state=42)
```

จากตัวอย่างข้างต้น ในการสร้างโมเดลนั้น เป็นการใช้ค่าเบื้องต้นในการสร้างโมเดล ดังนั้นประสิทธิภาพอาจจะไม่ดีมาก ดังนั้น ควรที่จะปรับเปลี่ยนพารามิเตอร์ (Tuning Parameter) เพื่อให้โมเดลที่สร้างนั้นมีประสิทธิภาพดีที่สุด สามารถทำได้โดยใช้คำสั่ง GridSearchCV()

```
from sklearn.grid_search import GridSearchCV

param_grid = {'svc__C': [1, 5, 10, 50],
              'svc__gamma': [0.0001, 0.0005, 0.001, 0.005]}
grid = GridSearchCV(model, param_grid)

%time grid.fit(Xtrain, ytrain)
print(grid.best_params_)
```

```
CPU times: user 1min 14s, sys: 1min, total: 2min 15s
Wall time: 33.8 s
{'svc__gamma': 0.001, 'svc__C': 5}
```

ค่าพารามิเตอร์ที่ใช้ในการปรับเปลี่ยนได้แก่ค่า **svc__C** และ **svc__gamma** ซึ่งเป็นค่าพารามิเตอร์ของ RBF Kernel โดยค่าที่ดีที่สุดในการทดสอบคือ **svc__gamma = 0.001** และ **svc__c = 5** จากนั้นนำค่าทั้ง 2 ค่าไปสร้างโมเดล ดังนี้

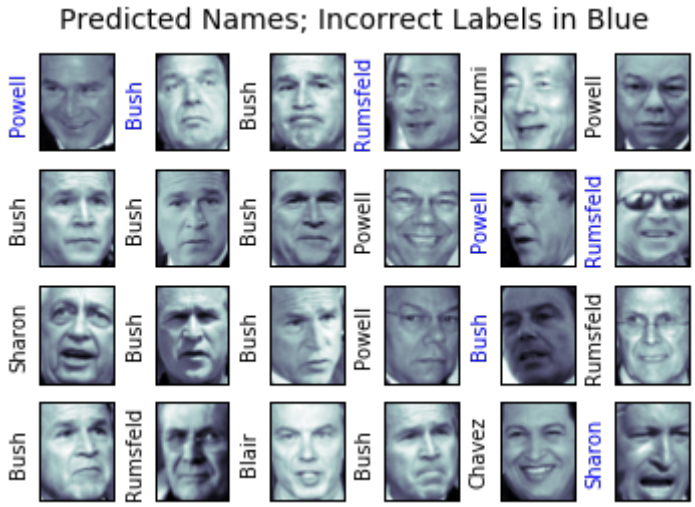
```
model = grid.best_estimator_
```

โมเดลที่สร้างด้วยตัวแปรที่ดีที่สุดจะถูกจัดเก็บไว้ที่ตัวแปรชื่อว่า model จากนั้นจึงสามารถนำไปเพื่อพยากรณ์ข้อมูลชุดทดสอบ ดังตัวอย่างต่อไปนี้

```
yfit = model.predict(Xtest)
```

โดยผลลัพธ์ที่ได้จากการพยากรณ์จะถูกจัดเก็บไว้ที่ตัวแปร yfit จากนั้นสามารถ Visualization รูปภาพใบหน้า และแสดงคำตอบในการพยากรณ์ ดังต่อไปนี้

```
fig, ax = plt.subplots(4, 6)

for i, axi in enumerate(ax.flat):
     axi.imshow(Xtest[i].reshape(62, 47), cmap='bone')
     axi.set(xticks=[], yticks=[])
     axi.set_ylabel(faces.target_names[yfit[i]].split()[-1],
               color='black' if yfit[i] == ytest[i]
               else 'blue')
     fig.suptitle('Predicted Names; Incorrect Labels in Blue',
size=14);
```



ภาพประกอบที่ 18: ผลลัพธ์ที่ได้จากการพยากรณ์ใบหน้าด้วยวิธี PCA และ SVM

จากภาพประกอบที่ 18 รายชื่อสีน้ำเงิน คือการพยากรณ์ที่ผิดพลาด และรายชื่อสีดำคือการพยากรณ์ที่ถูกต้อง

การวัดประสิทธิภาพของการรู้จำใบหน้า (Classification Report)

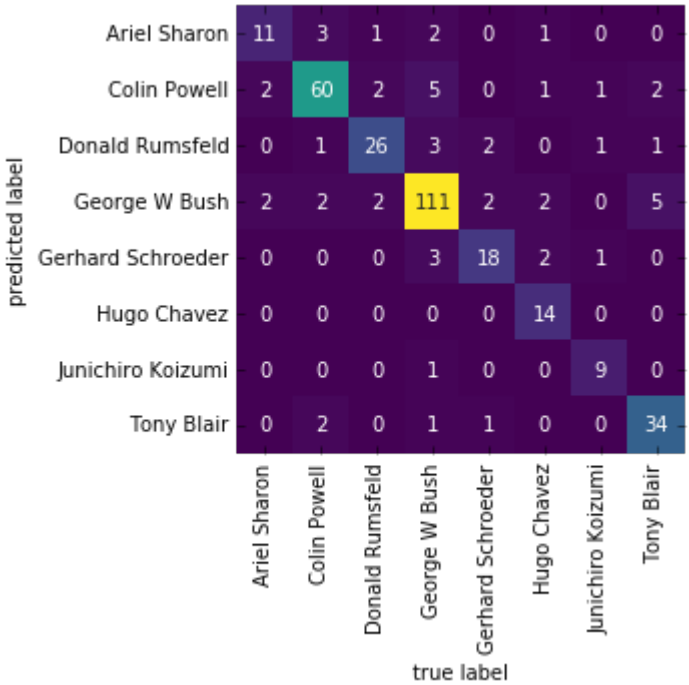
การวัดประสิทธิภาพของการรู้จำใบหน้า ในกรณีนี้ใช้คำสั่ง classification_report() เพื่อที่จะตรวจสอบค่า precision, recall และ f1-score

```
from sklearn.metrics import classification_report

print(classification_report(ytest, yfit,
                            target_names=faces.target_names))
```

```
                   precision    recall  f1-score   support

     Ariel Sharon       0.61      0.73      0.67        15
     Colin Powell       0.82      0.88      0.85        68
  Donald Rumsfeld       0.76      0.84      0.80        31
    George W Bush       0.88      0.88      0.88       126
Gerhard Schroeder       0.75      0.78      0.77        23
      Hugo Chavez       1.00      0.70      0.82        20
Junichiro Koizumi       0.90      0.75      0.82        12
       Tony Blair       0.89      0.81      0.85        42

      avg / total       0.85      0.84      0.84       337
```

การแสดงความถูกต้องของการพยากรณ์ด้วย Confusion Matrix

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

mat = confusion_matrix(ytest, yfit)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            cmap='viridis',
            xticklabels=faces.target_names,
            yticklabels=faces.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



อัตราความถูกต้อง (Accuracy Result) ของการพยากรณ์รูปภาพใบหน้า

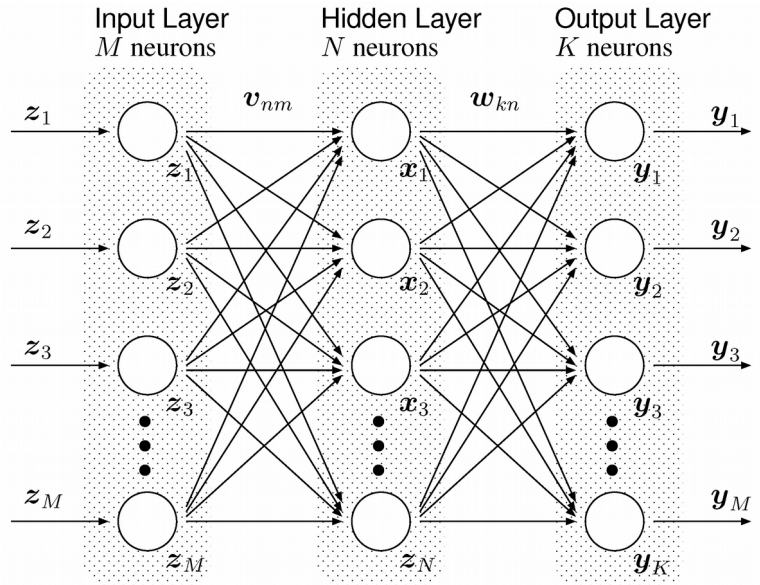
การคำนวณความถูกต้อง (Accuracy Result) ของอัลกอริธึม SVM สามารถใช้คำสั่ง accuracy_score() โดยผลลัพธ์ที่ได้จากการพยากรณ์คือ 83.97%

```
from sklearn.metrics import accuracy_score

print("Accuracy", accuracy_score(ytest, yfit)*100)
```

```
('Accuracy', 83.97626112759644)
```

บทที่ 13
การรู้จำตัวอักษร
(Character Recognition)

การรู้จำตัวอักษร (Character Recognition) ด้วยนิวรอลเน็ตเวิร์ก แบบ Multi-Layer Perceptron (MLP) ในกรณีนี้ทดสอบกับชุดข้อมูล MNIST โดยคุณลักษณะพิเศษ (Feature Extraction) ที่นำมาใช้คือค่าสีเทาของแต่ละพิกเซล ดังนั้น รูปภาพขนาด 28x28 พิกเซลจะมีคุณลักษณะพิเศษจำนวน 784 attribute ที่จะนำไปเรียนรู้



ภาพประกอบที่ 19: ตัวอย่างโครงสร้างของ Multi-Layer Perceptron (MLP)

จากภาพประกอบที่ 19 โครงสร้างของนิวรอลเน็ตเวิร์กแบบ MLP ประกอบด้วย 3 ชั้น (Layer) คือ Input Layer, Hidden Layer และ Output Layer โดยทั้ง 3 ชั้นจะมีการเชื่อมต่อกันของโหนดอย่างสมบูรณ์ (Fully-Connected)

เมื่อนำมาปรับใช้กับข้อมูลรูปภาพตัวอักษรที่มีขนาด 28x28 พิกเซล ดังนั้น ในชั้น Input Layer จึงกำหนดให้มีจำนวน 784 โหนด และในชั้น Output Layer ถูกกำหนดให้เป็น 10 โหนด

ตามจำนวนของตัวเลข 0-9 ส่วนในชั้น Hidden Layer นั้น จะต้องทำการคำนวณเพื่อหาจำนวนโหนดที่เหมาะสมที่สุดกับชุดข้อมูล

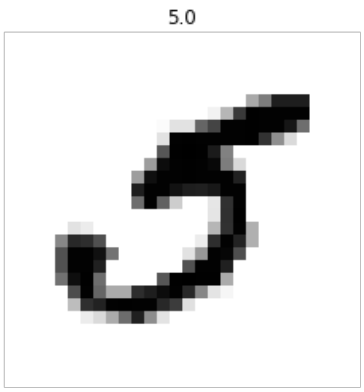
ในกรณีนี้ใช้ชุดข้อมูล MNIST ที่มีจำนวน 70,000 ตัวเลข โดยใช้โปรแกรม scipy ในการโหลด

```
from scipy.io import loadmat

mnist_raw = loadmat("mldata/mnist-original.mat")
mnist = {
    "data": mnist_raw["data"].T,
    "target": mnist_raw["label"][0],
    "COL_NAMES": ["label", "data"],
    "DESCR": "mldata.org dataset: mnist-original",
    }

X,y = mnist['data'], mnist['target']
X.shape, y.shape
```

```
((70000, 784), (70000,))
```

จากนั้นใช้คำสั่งดังต่อไปนี้เพื่อ Visualization รูปภาพตัวเลข

```
import matplotlib.pyplot as plt

some_digit = X[36000]
some_digit_image = some_digit.reshape(28, 28)

plt.imshow(
    some_digit_image,
    cmap = plt.cm.binary,
    interpolation="nearest")

plt.title(y[36000])
plt.axis("off")
plt.show()
```



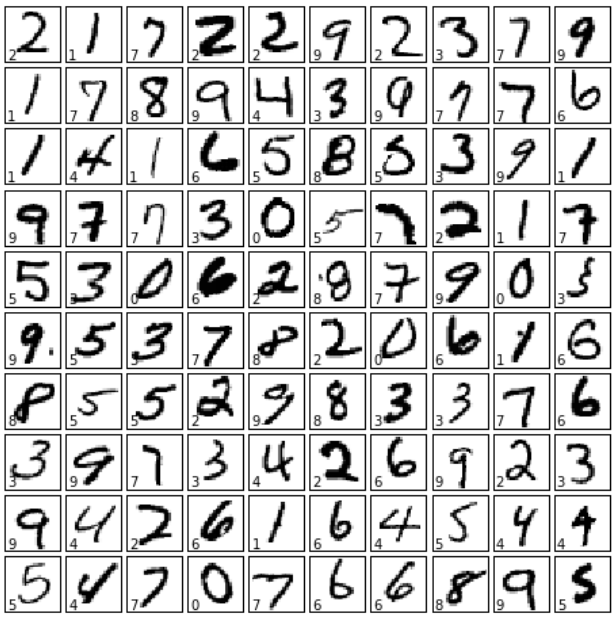
เนื่องจากในชุดข้อมูลดังกล่าว ตั้งแต่ชุดข้อมูลที่ 1-60,000 และชุดข้อมูลที่ 60,001-70,000 ข้อมูลได้ถูกจัดเรียงตามตัวเลข 0-9 ดังนั้น หากต้องการข้อมูลบางส่วนมาเพื่อทดสอบ จึงควรที่จะสลับ (Shuffle) ข้อมูลเสียก่อน การสลับข้อมูลสามารถทำได้ดังต่อไปนี้

```
import numpy as np

shuffle_index = np.random.permutation(70000)
X, y = X[shuffle_index], y[shuffle_index]

X_train, X_test, y_train, y_test = X[:60000], X[60000:],
                                   y[:60000], y[60000:]
print(X_train.shape, X_test.shape, y_train.shape,\
     y_test.shape)
```

```
((60000, 784), (10000, 784), (60000,), (10000,))
```

จากนั้นสามารถแสดงข้อมูล โดยแสดงแบบ Visualization และแสดง Label ประกอบ ดังนี้

```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(10, 10, figsize=(8, 8),
                    subplot_kw={'xticks':[], 'yticks':[]},
                    gridspec_kw=dict(hspace=0.1, wspace=0.1))

for i, ax in enumerate(axes.flat):
    ax.imshow(X_train[i].reshape((28,28)), cmap='binary',
               interpolation='nearest')
    ax.text(0.05, 0.05, str(int(y_train[i])),
               transform=ax.transAxes, color='black')

plt.show()
```



สร้างโมเดลของอัลกอริธึม MLP

การใช้งาน MLP ใช้ scikit-learn จะต้องใช้โมดูล MLPClassifier โดยในตัวอย่างได้กำหนดให้ Hidden Layer มีขนาด 100 โหนด (hidden_layer_sizes) และทำการทดสอบจำนวน 10 รอบ (max_iter) และกำหนดอัตราการเรียนรู้ที่ 0.001 (learning_rate_init) จากนั้นใช้คำสั่ง fit() เพื่อสร้างโมเดล และเก็บโมเดลไว้ที่ตัวแปร mlp

```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=10,
                    alpha=1e-4,
                    solver='sgd', verbose=10, tol=1e-5,
                    random_state=1,
                    learning_rate_init=0.001)

mlp.fit(X_train, y_train)
```

```
Iteration 1, loss = 1.81871326
Iteration 2, loss = 1.05944149
Iteration 3, loss = 0.69554161
Iteration 4, loss = 0.50766222
Iteration 5, loss = 0.38525626
Iteration 6, loss = 0.33714014
Iteration 7, loss = 0.30388212
Iteration 8, loss = 0.28367218
Iteration 9, loss = 0.26573927
Iteration 10, loss = 0.25603315

MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=10, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=1, shuffle=True,
       solver='sgd', tol=1e-05, validation_fraction=0.1, verbose=10,
       warm_start=False)
```

การวัดประสิทธิภาพของการเรียนรู้

ในกรณีนี้ ใช้คำสั่ง score() เพื่อดูผลการทดลองกับข้อมูลชุดเรียนรู้ และข้อมูลชุดทดสอบ โดยใช้คำสั่งดังต่อไปนี้

```
print("Training set score: %f" % mlp.score(X_train, y_train))
print("Test set score: %f" % mlp.score(X_test, y_test))
```

การพยากรณ์และวัดประสิทธิภาพของการู้จำ

ในการพยากรณ์ให้ใช้คำสั่ง predict() เพื่อพยากรณ์ข้อมูล X_test ดังตัวอย่างต่อไปนี้

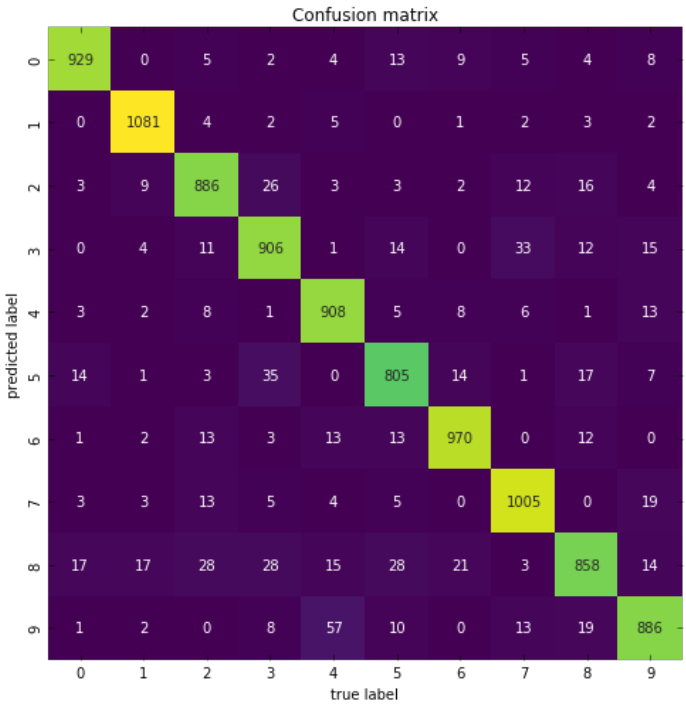
```
yfit = mlp.predict(X_test)
```

```
from sklearn.metrics import classification_report

tn = ['0','1','2','3','4','5','6','7','8','9']
print(classification_report(y_test, yfit,
                            target_names=tn))
```

```
             precision    recall  f1-score   support

          0       0.95      0.96      0.95       971
          1       0.98      0.96      0.97      1121
          2       0.92      0.91      0.92       971
          3       0.91      0.89      0.90      1016
          4       0.95      0.90      0.92      1010
          5       0.90      0.90      0.90       896
          6       0.94      0.95      0.95      1025
          7       0.95      0.93      0.94      1080
          8       0.83      0.91      0.87       942
          9       0.89      0.92      0.90       968

avg / total       0.92      0.92      0.92     10000
```

แสดงผลลัพธ์จากการพยากรณ์ด้วย Confusion Matrix

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

mat = confusion_matrix(y_test, yfit)

fig, ax = plt.subplots(figsize=(8,8))
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False, cmap='viridis',
            xticklabels=tn,
            yticklabels=tn,
            ax=ax)
plt.title('Confusion matrix')
plt.xlabel('true label')
plt.ylabel('predicted label');
```



แสดงอัตราความถูกต้องจากการพยากรณ์ โดยผลลัพธ์ที่ได้จากการพยากรณ์คือ 92.34%

```
from sklearn.metrics import accuracy_score

print("Accuracy", accuracy_score(y_test, yfit)*100)
```

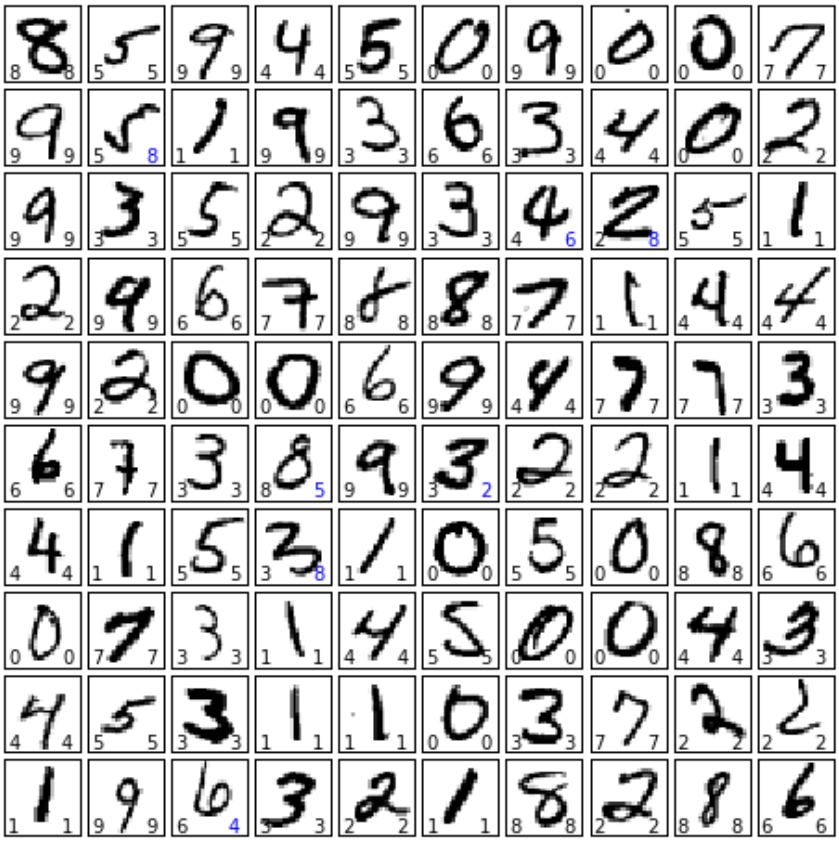
```
('Accuracy', 92.34)
```

การ Visualization รูปภาพตัวเลข และแสดงผลการพยากรณ์

เพื่อให้สะดวกต่อการเปรียบเทียบระหว่างค่าที่แท้จริง และค่าที่พยากรณ์ สามารถ Visualization ข้อมูลได้ดังต่อไปนี้

```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(10, 10, figsize=(8, 8),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1, wspace=0.1))

for i, ax in enumerate(axes.flat):
    ax.imshow(X_test[i].reshape((28,28)), cmap='binary',
              interpolation='nearest')
```

```
        # actual class
        ax.text(0.05, 0.05, str(int(y_test[i])),
                transform=ax.transAxes,
                color='black')

       # predict class
       ax.text(0.75, 0.05, str(int(yfit[i])),
               transform=ax.transAxes,
               color='black' if yfit[i] == y_test[i] else 'blue')

plt.show()
```



ภาพประกอบที่ 20: เปรียบเทียบระหว่างค่าที่แท้จริง และค่าที่ได้จากการพยากรณ์

จากภาพประกอบที่ 20 ในแต่ละรูปจะมีตัวเลขกำกับ 2 ตัวเลข โดยตัวเลขทางด้านซ้ายมือคือ Label ของตัวเลขนั้น ๆ ส่วนตัวเลขทางด้านขวาคือผลลัพธ์ที่ได้จากการพยากรณ์ ดังนั้น หากเป็นการพยากรณ์ที่ผิดพลาดจะแสดงเป็นสีน้ำเงิน แต่หากพยากรณ์ถูกต้องก็จะแสดงเป็นสีดำ