# Hands-On Artificial Intelligence for IoT: Expert machine learning and deep learning techniques for developing smarter IoT systems

Amita Kapoor
NePeur

# Hands-On Artificial Intelligence for IoT

Expert machine learning and deep learning techniques for developing smarter IoT systems

**Amita Kapoor**

Packt

BIRMINGHAM - MUMBAI

# Hands-On Artificial Intelligence for IoT

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

**Commissioning Editor:** Amey Varangaonkar
**Acquisition Editor:** Nelson Morris
**Content Development Editor:** Karan Thakkar
**Technical Editor:** Adya Anand
**Copy Editor:** Safis Editing
**Project Coordinator:** Hardik Bhinde
**Proofreader:** Safis Editing
**Indexer:** Tejal Daruwale Soni
**Graphics:** Jisha Chirayil
**Production Coordinator:** Arvindkumar Gupta

First published: January 2019

Production reference: 1131015

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78883-606-7

www.packtpub.com

# Dedication

*To my friend and mentor Narotam Singh for being my gradient ascent in the dataset called life.*
*A part of my royalties will go to smilefoundation.org, a non-profit organization based in India*
*working on welfare projects on education, healthcare, livelihood, and the*
*empowerment of women in remote villages and slums across the different state of India.*

*– Amita Kapoor*

# Contributors

## About the author

**Amita Kapoor** is an associate professor in the Department of Electronics, SRCASW, University of Delhi, and has been actively teaching neural networks and artificial intelligence for the last 20 years. She completed her master's in electronics in 1996 and her PhD in 2011. During her PhD she was awarded the prestigious DAAD fellowship to pursue part of her research at the Karlsruhe Institute of Technology, Karlsruhe, Germany. She was awarded the Best Presentation Award at the Photonics 2008 international conference. She is an active member of ACM, AAAI, IEEE, and INNS. She has co-authored two books. She has more than 40 publications in international journals and conferences. Her present research areas include machine learning, artificial intelligence, deep reinforcement learning, and robotics.

*I would like to thank Prof Ajit Jaokar, University of Oxford; his IoT course was the inspiration behind this book. Special thanks to Erin LeDell, Chief Machine Learning Scientist at H2O.ai for her thoughtful suggestions. I would also like to thank Armando Fandango, Narotam Singh, Ruben Olivas, and Hector Velarde for their input.*
*I am grateful for the support of my colleagues and students. Last but not least, I would like to thank the entire Packt team, with a special mention to Tushar Gupta, Karan Thakkar, and Adya Anand for their continuous motivation.*

# About the reviewers

**Hector Duran Lopez Velarde** received a B.Che.E. from UPAEP and an MSc in automation and artificial intelligence from Tecnologico de Monterrey ITESM, Mexico, in 2000. He has worked as a controls and automation engineer for companies such as Honeywell and General Electric, among others. He also has participated in several research projects as a technical lead. His experience in software development, process simulation, artificial intelligence, and industrial automation has led him to the current development of complete IoT solutions in the automotive, textile, and pharmaceutical industries. He is currently working on a research center of IoT.

> *Huge thanks to my wife, Yaz, and to my children, Ivana and Hector, for all their support and love.*

**Ruben Oliva Ramos** is a computer engineer from Tecnologico of León Institute, with a master's degree in computer and electronics systems engineering with a networking specialization from the University of Salle Bajio. He has more than 5 years' experience of developing web apps to control and monitor devices connected to Arduino and Raspberry Pi, using web frameworks and cloud services to build IoT applications. He has authored *Raspberry Pi 3 Home Automation Projects*, *Internet of Things Programming with JavaScript*, *Advanced Analytics with R and Tableau*, and *SciPy Recipes* for Packt.

> *I would like to thank my savior and lord, Jesus Christ for giving me strength and courage to pursue this project, to my dearest wife, Mayte, our two lovely sons, Ruben and Dario, To my dear father (Ruben), my dearest mom (Rosalia), my brother (Juan Tomas), and my sister (Rosalia) whom I love. I'm very grateful with Pack Publishing for giving the opportunity to collaborate as an author and reviewer, to belong to this honest and professional team.*

# Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit `authors.packtpub.com` and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

# Table of Contents

# Preface

The mission of this book is to enable the reader to build AI-enabled IoT applications. With the surge in IoT devices, there are many applications that use data science and analytics to utilize the terabyte of data generated. However, these applications do not address the challenge of continually discovering patterns in IoT data. In this book, we cover the various aspects of AI theory and implementation that the reader can utilize to make their IoT solutions smarter by implementing AI techniques.

The reader starts by learning the basics of AI and IoT devices and how to read IoT data from various sources and streams. Then we introduce various ways to implement AI with examples in TensorFlow, scikit learn, and Keras. The topics covered include machine learning, deep learning, genetic algorithms, reinforcement learning, and generative adversarial networks. We also show the reader how to implement AI using distributed technologies and on the cloud. Once the reader is familiar with AI techniques, then we introduce various techniques for different kinds of data generated and consumed by IoT devices, such as time series, images, audio, video, text, and speech.

After explaining various AI techniques on various kinds of IoT data, finally, we share some case studies with the reader from the four major categories of IoT solutions: personal IoT, home IoT, industrial IoT, and smart city IoT.

## Who this book is for

The audience for this book is anyone who has a basic knowledge of developing IoT applications and Python and wants to make their IoT applications smarter by applying AI techniques. This audience may include the following people:

- IoT practitioners who already know how to build IoT systems, but now they want to implement AI to make their IoT solution smart.

- Data science practitioners who have been building analytics with IoT platforms, but now they want to transition from IoT analytics to IoT AI, thus making their IoT solutions smarter.

- Software engineers who want to develop AI-based solutions for smart IoT devices.

- Embedded system engineers looking to bring smartness and intelligence to their products.

# What this book covers

`Chapter 1`, *Principles and Foundations of IoT and AI*, introduces the basic concepts IoT, AI, and data science. We end the chapter with an introduction to the tools and datasets we will be using in the book.

`Chapter 2`, *Data Access and Distributed Processing for IoT*, covers various methods of accessing data from various data sources, such as files, databases, distributed data stores, and streaming data.

`Chapter 3`, *Machine Learning for IoT*, covers the various aspects of machine learning, such as supervised, unsupervised, and reinforcement learning for IoT. The chapter ends with tips and tricks to improve your models' performance.

`Chapter 4`, *Deep Learning for IoT*, explores the various aspects of deep learning, such as MLP, CNN, RNN, and autoencoders for IoT. It also introduces various frameworks for deep learning.

`Chapter 5`, *Genetic Algorithms for IoT*, discusses optimization and different evolutionary techniques employed for optimization with an emphasis on genetic algorithms.

`Chapter 6`, *Reinforcement Learning for IoT*, introduces the concepts of reinforcement learning, such as policy gradients and Q-networks. We cover how to implement deep Q networks using TensorFlow and learn some cool real-world problems where reinforcement learning can be applied.

`Chapter 7`, *Generative Models for IoT*, introduces the concepts of adversarial and generative learning. We cover how to implement GAN, DCGAN, and CycleGAN using TensorFlow, and also look at their real-life applications.

`Chapter 8`, *Distributed AI for IoT*, covers how to leverage machine learning in distributed mode for IoT applications.

`Chapter 9`, *Personal and Home and IoT*, goes over some exciting personal and home applications of IoT.

`Chapter 10`, *AI for Industrial IoT*, explains how to apply the concepts learned in this book to two case studies with industrial IoT data.

`Chapter 11`, *AI for Smart Cities IoT*, explains how to apply the concepts learned in this book to IoT data generated from smart cities.

`Chapter 12`, *Combining It All Together*, covers how to pre-process textual, image, video, and audio data before feeding it to models. It also introduces time series data.

# To get the most out of this book

To get the most out of this book, download the examples code from the GitHub repository and practice with the Jupyter Notebooks provided.

## Download the example code files

You can download the example code files for this book from your account at `www.packtpub.com`. If you purchased this book elsewhere, you can visit `www.packtpub.com/support` and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at `www.packtpub.com`.
2. Select the **SUPPORT** tab.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box and follow the onscreen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR/7-Zip for Windows
- Zipeg/iZip/UnRarX for Mac
- 7-Zip/PeaZip for Linux

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/Hands-On-Artificial-Intelligence-for-IoT`. We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

## Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here:

`http://www.packtpub.com/sites/default/files/downloads/9781788836067_ColorImages.pdf`.

## Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "This declares two placeholders with the names `A` and `B`; the arguments to the `tf.placeholder` method specify that the placeholders are of the `float32` datatype."

A block of code is set as follows:

```
# Declare placeholders for the two matrices
A = tf.placeholder(tf.float32, None, name='A')
B = tf.placeholder(tf.float32, None, name='B')
```

**Bold**: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "At the bottom of the stack, we have the device layer, also called the **perception layer**."

Warnings or important notes appear like this.

Tips and tricks appear like this.

# Get in touch

Feedback from our readers is always welcome.

**General feedback**: Email `feedback@packtpub.com` and mention the book title in the subject of your message. If you have questions about any aspect of this book, please email us at `questions@packtpub.com`.

**Errata**: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/submit-errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

**Piracy**: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packtpub.com` with a link to the material.

**If you are interested in becoming an author**: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

# Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packtpub.com`.

# 1
# Principles and Foundations of IoT and AI

Congratulations on purchasing this book; it suggests that you're keenly interested in keeping yourself updated with the recent advancements in technology. This book deals with the three big trends in the current business scenario, **Internet of Things** (**IoT**), big data, and **Artificial Intelligence** (**AI**). The exponential growth of the number of devices connected to the internet and the exponential volume of data created by them necessitates the use of the analytical and predictive techniques of AI and **deep learning** (**DL**). This book specifically targets the third component, the various analytical and predictive methods or models available in the field of AI for the big data generated by IoT.

This chapter will briefly introduce you to these three trends and will expand on how they're interdependent. The data generated by IoT devices is uploaded to the cloud, hence you'll also be introduced to the various IoT cloud platforms and the data services they offer.

This chapter will cover the following points:

- Knowing what's a thing in IoT, what devices constitute things, what're the different IoT platforms, and what's an IoT vertical
- Knowing what's big data and understanding how the amount of data generated by IoT lies in the range of big data
- Understanding how and why AI can be useful for making sense of the voluminous data generated by IoT
- With the help of an illustration, understanding how IoT, big data, and AI together can help shape a better world
- Learning about some of the tools that would be needed to perform the analysis

## What is IoT 101?

The term IoT was coined by Kevin Ashton in 1999. At that time, most of the data fed to computers were generated by humans; he proposed that the best way would be for computers to take data directly, without any intervention from humans. And so he proposed things such as RFID and sensors, which will gather data, should be connected to the network and feed directly to the computer.

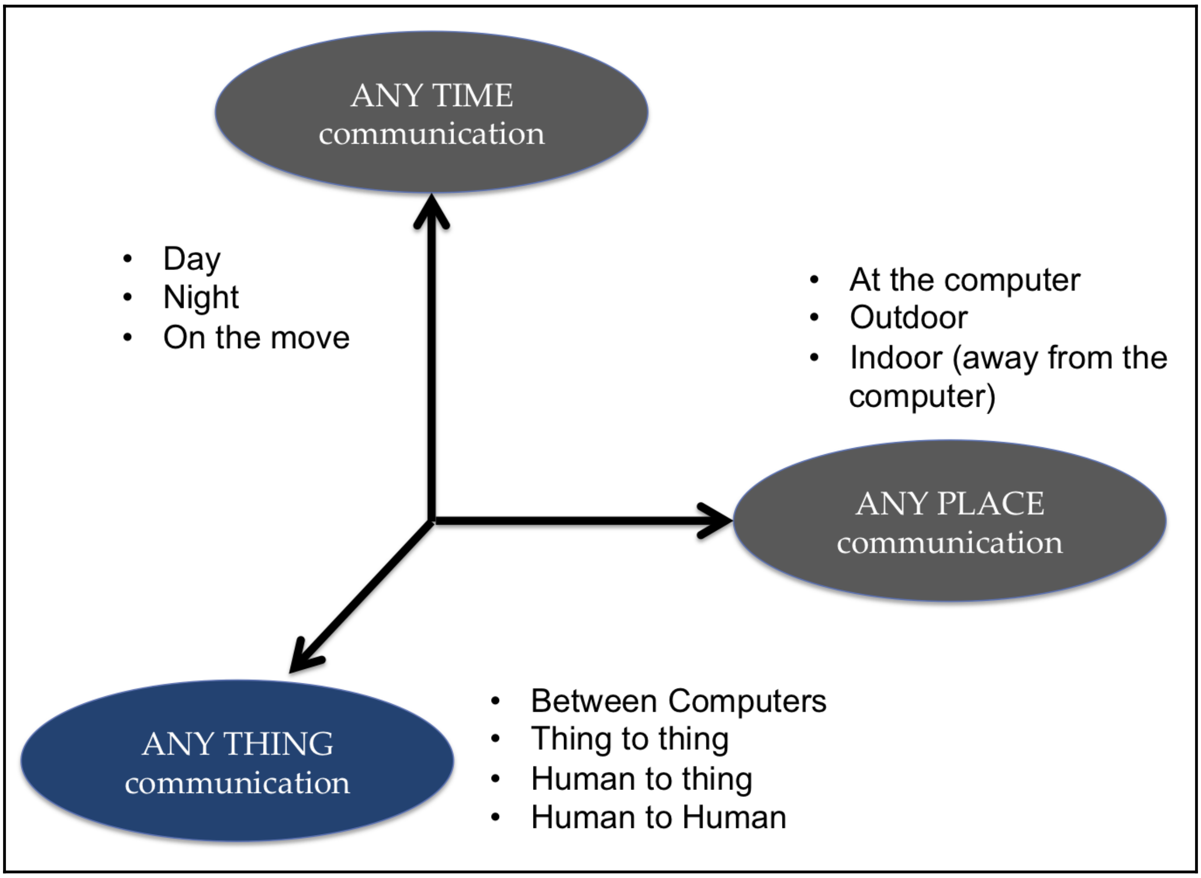
TIP

You can read the complete article where Ashton talks about what he means by IoT here: `http://www.itrco.jp/libraries/RFIDjournal-That%20Internet%20of%20Things%20Thing.pdf`.

Today IoT, (also called the **internet of everything** and sometimes fog network), refers to a wide range of things such as sensors, actuators, and smartphones connected to the internet. These things can be anything: a person with a wearable device (or even mobile phone), an RFID-tagged animal, or even our day-to-day devices such as a refrigerator, washing machine, or even a coffee machine. These things can be physical things—that is, things that exist in the physical world and can be sensed, actuated, and connected—or of the information world (virtual thing)—that is, things that aren't tangibly present but exist as information (data) and can be stored, processed, and accessed. These things necessarily have the ability to communicate directly with the internet; optionally, they might have the potentiality of sensing, actuation, data capture, data storage, and data processing.

The **International Telecommunication Unit** (**ITU**), a United Nations agency, defines IoT as: *a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies*. You can learn more at `https://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx`.

The wide expanse of ICT already provided us with communication at any time or any place, the IoT added a new dimension **ANY THING communication**:



New dimension introduced in IoT (adapted from b-ITU-T Y.2060 report)

It's predicted that IoT as technology will have a far-reaching impact on people and the society we live in. To give you a glimpse of its far-reaching effects, consider the following scenarios:

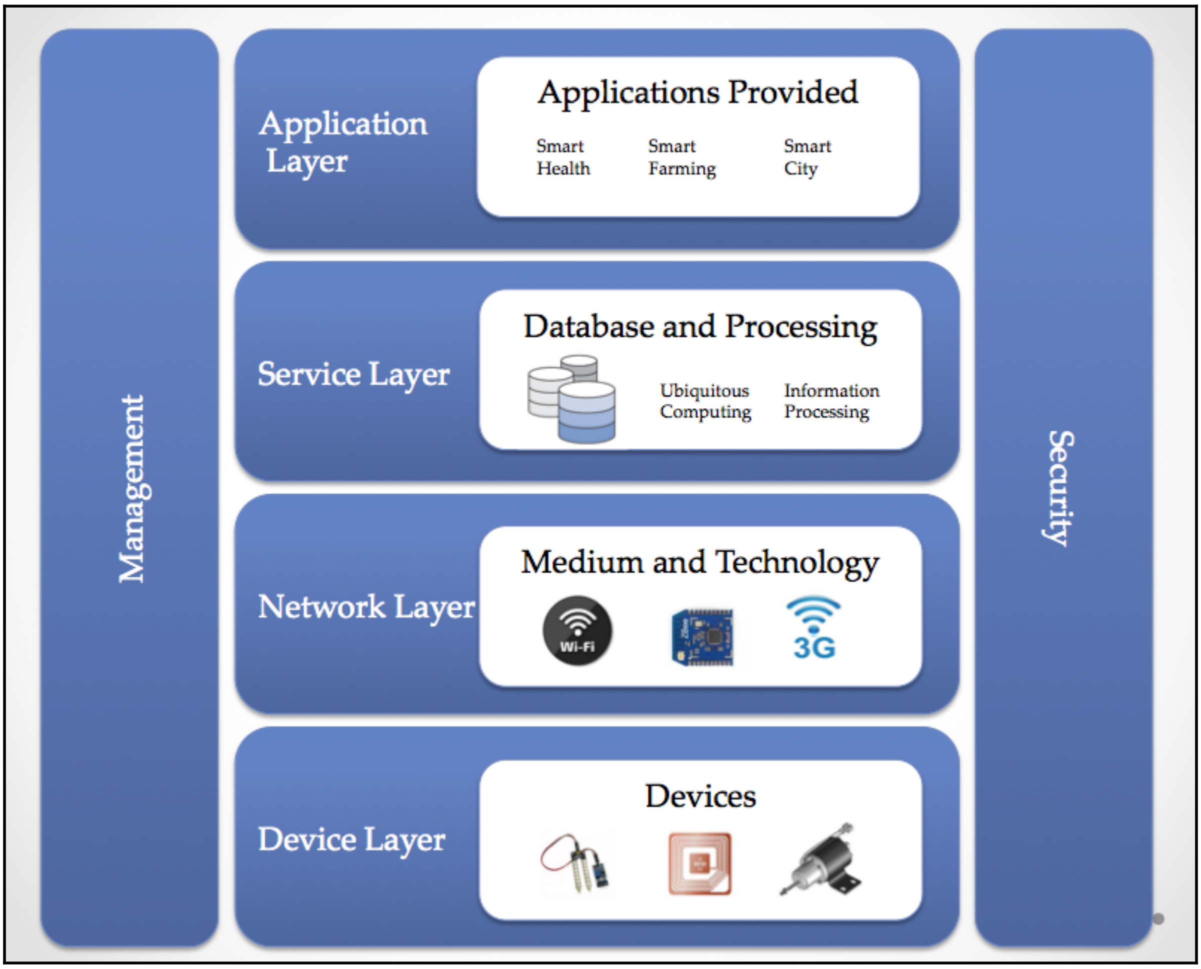
- You, like me, live in a high rise building and are very fond of plants. With lots of effort and care, you've made a small indoor garden of your own using potted plants. Your boss asks you to go for a week-long trip, and you're worried your plants won't survive for a week without water. The IoT solution is to add soil moisture sensors to your plants, connect them to the internet, and add actuators to remotely switch on or off the water supply and artificial sunlight. Now, you can be anywhere in the world, but your plants won't die, and you can check the individual plant's soil moisture condition and water it as needed.
- You had a very tiring day at the office; you just want to go home and have

someone make you coffee, prepare your bed, and heat up water for a bath, but sadly you're home alone. Not anymore; IoT can help. Your IoT-enabled home assistant can prepare the right flavor coffee from the coffee machine, order your smart water heater to switch on and maintain the water temperature exactly the way you want, and ask your smart air conditioner to switch on and cool the room.

The choices are limited only by your imagination. The two preceding scenarios correspond to consumer IoT—the IoT with focus on consumer-oriented applications. There also exists a large scope of **Industry IoT** (**IIoT**) where manufacturers and industries optimize processes and implement remote monitoring capabilities to increase productivity and efficiency. In this book, you'll find the hands-on experience with both IoT applications.

# IoT reference model

Just like the OSI reference model for the internet, IoT architecture is defined through six layers: four horizontal layers and two vertical layers. The two vertical layers are **Management** and **Security** and they're spread over all four horizontal layers, as seen in the following diagram:



IoT layers

**Device Layer**: At the bottom of the stack, we have the device layer, also called the **perception layer**. This layer contains the physical things needed to sense or control the physical world and acquire data (perceive the physical world). The existing hardware such as sensors, RFID, and actuators constitutes the perception layer.

**Network Layer**: This layer provides the networking support and transfer of data over either wired or wireless network. The layer securely transmits the information from the devices in the device layer to the information processing system. Both transmission **Medium and Technology** are part of the networking layer. Examples include 3G, UMTS, ZigBee, Bluetooth, Wi-Fi, and so on.

**Service Layer**: This layer is responsible for service management. It receives information from the network layer, stores it into the database, processes that information, and can make an automatic decision based on the results.

**Application Layer**: This layer manages the applications dependent upon the information processed in the service layer. There's a wide range of applications that can be implemented by IoT: smart cities, smart farming, and smart homes, to name a few.

# IoT platforms

Information from the network layer is often managed with the help of IoT platforms. Many companies today provide IoT platform services, where they help not only with data but also enable seamless integration with different hardware. Since it functions as a mediator between the hardware and application layer, IoT platforms are also referred to as IoT middleware and are part of the service layer in the IoT reference stack. IoT platforms provide the ability to connect and communicate with things from anywhere in the world. In this book, we'll briefly cover some popular IoT platforms such as the Google Cloud Platform, Azure IoT, Amazon AWS IoT, Predix, and H2O.

You can select which IoT platform is best for you based on the following criteria:

- **Scalability**: Addition and deletion of new devices to the existing IoT network should be possible
- **Ease of use**: The system should be perfectly working and delivering all its specifications with minimum intervention
- **Third party integration**: Heterogeneous devices and protocols should be able to inter-network with each other
- **Deployment options**: It should be workable on a broad variety of hardware devices and software platforms
- **Data security**: The security of data and devices is ensured

# IoT verticals

A vertical market is a market in which vendors offer goods and services specific to an industry, trade, profession, or other groups of customers with specialized needs. IoT enables the possibility of many such verticals, and some of the top IoT verticals are as follows:

- **Smart building**: Buildings with IoT technologies can help in not only reducing the consumption of resources but also improving the satisfaction of the humans living or working in them. The buildings have smart sensors that not only monitors resource consumption but can also proactively detect residents' needs. Data is collected via these smart devices and sensors to remotely monitor a property's (buildings) energy, security, landscaping, HVAC, lighting, and so on. The data is then used to predict actions, which can be automated according to events and hence efficiency can be optimized, saving time, resources, and cost.

- **Smart agriculture**: IoT can enable local and commercial farming to be more environmentally friendly, cost-effective, and production efficient. Sensors placed through the farm can help in automating the process of irrigation. It's predicted that smart agricultural practices will enable a manifold increase in productivity and hence food resources.
- **Smart city**: A smart city can be a city with smart parking, a smart mass transit system, and so on. A smart city has the capability to address traffic, public safety, energy management, and more for both its government and citizens. By using advanced IoT technologies, it can optimize the usage of the city infrastructure and quality of life for its citizens.
- **Connected healthcare**: IoT enables critical business and patient monitoring decisions to be made remotely and in real time. Individuals carry medical sensors to monitor body parameters such as heartbeat, body temperature, glucose level, and so on. The wearable sensors such as accelerometers and gyroscopes can be used to monitor a person's daily activity.

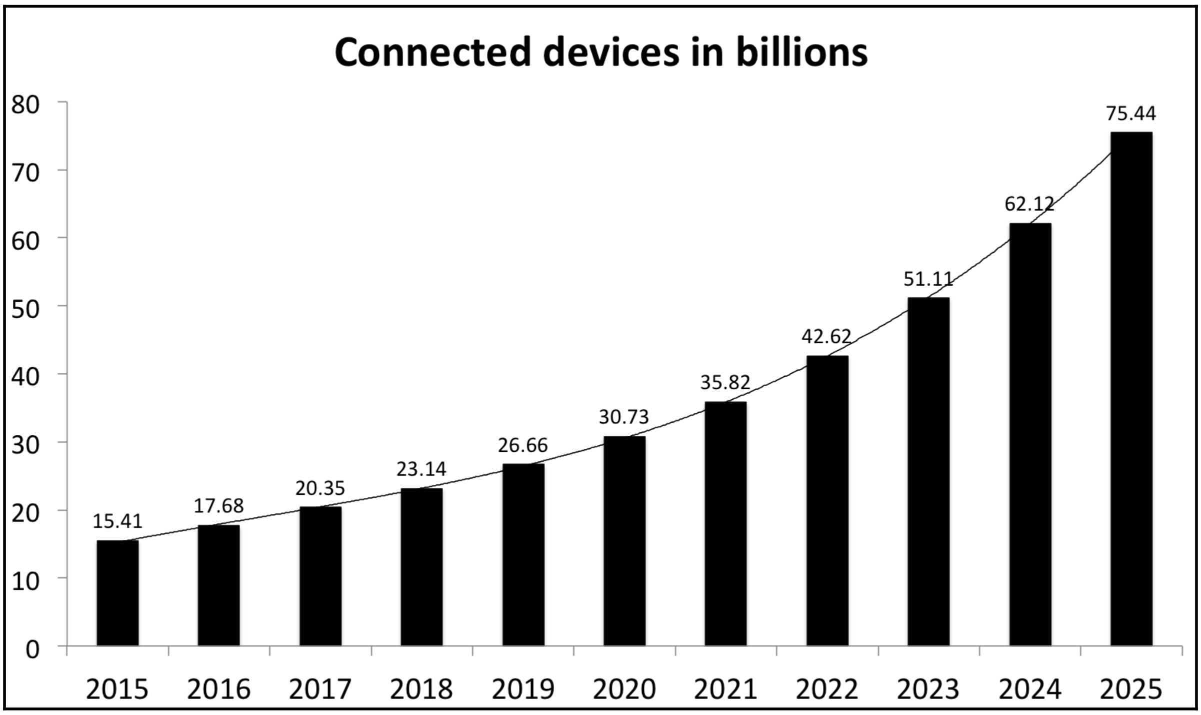
We'll be covering some of them as a case study in this book. The content of this book is focused on the information processing and applications being implemented by IoT and so we'll not be going into details of the devices, architecture, and protocols involved in IoT reference stacks any further.

The interested reader can refer to the following references to know more about the IoT architecture and different protocols:

- Da Xu, Li, Wu He, and Shancang Li. *Internet of things in industries: A survey.* IEEE Transactions on industrial informatics 10.4 (2014): 2233-2243.
- Khan, Rafiullah, et al. *Future internet: The internet of things architecture, Possible Applications and Key Challenges.* **Frontiers of Information Technology** (**FIT**), 2012 10th International Conference on. IEEE, 2012.
- This website provides an overview of the protocols involved in IoT: `https://www.postscapes.com/internet-of-things-protocols/`.

# Big data and IoT

IoT has connected things never previously connected to the internet, such as car engines, resulting in the generation of a large amount of continuous data streams. The following screenshot shows explorative data by IHS of the number of connected devices in billions in future years. Their estimate shows that the number of IoT devices will reach **75.44** billion by **2025**:



Prediction about the growth of IoT devices by 2025

The full whitepaper, *IoT platforms: enabling the Internet of Things*, by IHS is available as PDF at: `https://cdn.ihs.com/www/pdf/enabling-IOT.pdf`.

The reduction in the sensor cost, efficient power consumption techniques, a large range of connectivity (infrared, NFC, Bluetooth, Wi-Fi, and so on), and the availability of cloud platforms that support IoT deployment and development are the major reasons for this pervasion of IoT in our homes, personal lives, and industry. This has also ignited companies to think about providing new services and develop new business models. Some examples include the following:

- **Airbnb**: It connects people so that they can rent out spare rooms and cottages to one another, and it earns the commission.
- **Uber**: It connects cab drivers with the travelers. The location of the traveler is used to assign them the nearest driver.

The amount of data generated in the process is both voluminous and complex, necessitating in big data. Big data and IoT are almost made for each other; the two work in conjunction.

Things are continuously generating an enormous amount of data streams that provide their statuses such as temperature, pollution level, geolocation, and proximity. The data generated is in time series format and is autocorrelated. The task becomes challenging because the data is dynamic in nature. Also, the data generated can be analyzed at the edge (sensor or gateway) or cloud. Before sending the data to the cloud, some form of IoT data transformation is performed. This may involve the following:

- Temporal or spatial analysis
- Summarizing the data at the edge
- Aggregation of data
- Correlating data in multiple IoT streams
- Cleaning data
- Filling in the missing values
- Normalizing the data
- Transforming it into different formats acceptable to the cloud

At the edge, **complex event processing** (**CEP**) is used to combine data from multiple sources and infer events or patterns.

The data is analyzed using stream analytics, for example, applying analytical tools to the stream of data, but developing the insights and rules used externally in an offline mode. The model is built offline and then applied to the stream of data generated. The data may be handled in different manners:

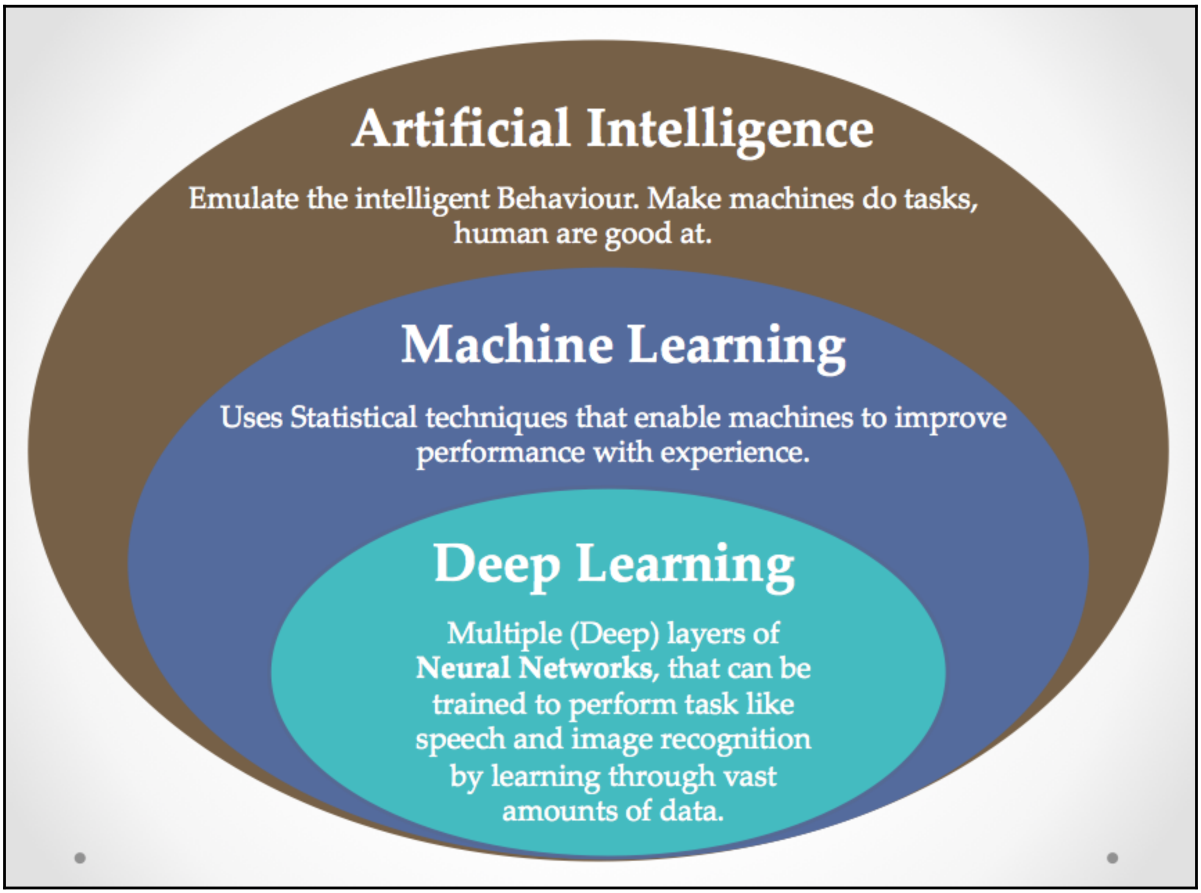
- **Atomic**: Single data at a time is used
- **Micro batching**: Group of data per batch
- **Windowing**: Data within a time frame per batch

The stream analytics can be combined with the CEP to combine events over a time frame and correlate patterns to detect special patterns (for example, anomaly or failure).

# Infusion of AI – data science in IoT

A very popular phrase among data scientists and machine learning engineers is *"AI is the new electricity"* said by Prof Andrew Ng in NIPS 2017, we can expand it, *If AI is the new electricity, Data is the new coal, and IoT the new coal-mine.*

IoT generates an enormous amount of data; presently, 90% of the data generated isn't even captured, and out of the 10% that is captured, most is time-dependent and loses its value within milliseconds. Manually monitoring this data continuously is both cumbersome and expensive. This necessitates a way to intelligently analyze and gain insight from this data; the tools and models of AI provide us with a way to do exactly this with minimum human intervention. The major focus of this book will be on understanding the various AI models and techniques that can be applied to IoT data. We'll be using both **machine learning** (**ML**) and DL algorithms. The following screenshot explains the relationship between **Artificial Intelligence**, **Machine Learning**, and **Deep Learning**:


Artificial Intelligence
Emulate the intelligent Behaviour. Make machines do tasks, human are good at.
Machine Learning
Uses Statistical techniques that enable machines to improve performance with experience.
Deep Learning
Multiple (Deep) layers of **Neural Networks**, that can be trained to perform task like speech and image recognition by learning through vast amounts of data.

AI, ML, and DL

By observing the behavior of multiple things, IoT with the help of big data and AI aims to gain insight into the data and optimize underlying processes. This involves multiple challenges:

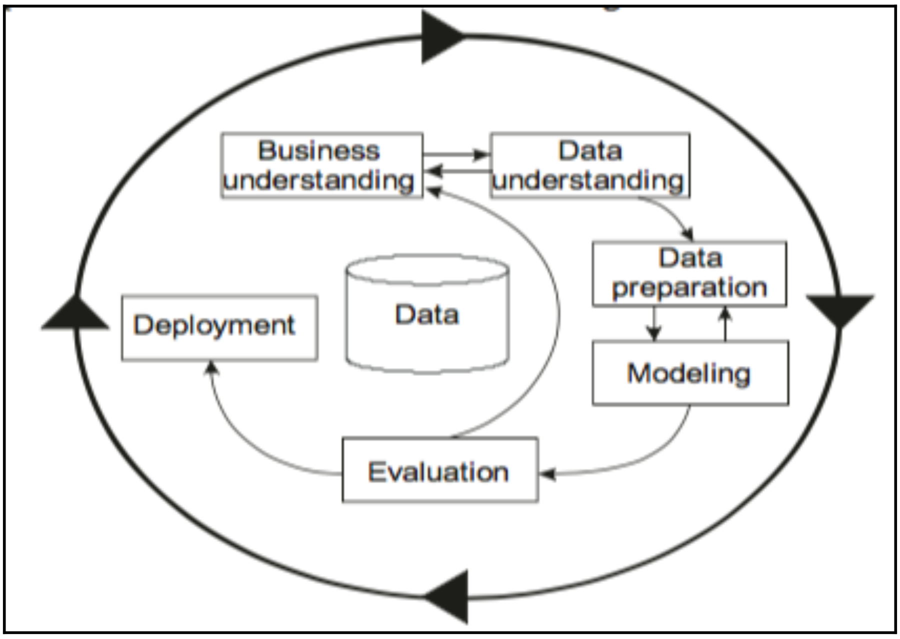
- Storing real-time generated events
- Running analytical queries over stored events
- Performing analytics using AI/ML/DL techniques over the data to gain insights and make predictions

# Cross-industry standard process for data mining

For IoT problems, the most used **data management** (**DM**) methodology is c**ross-industry standard process for data mining** (**CRISP-DM**) proposed by Chapman et al. It's a process model, which states the tasks that need to be carried out for successfully completing DM. It's a vendor-independent methodology divided into six different phases, such as the following:

1. **Business understanding**
2. **Data understanding**
3. **Data preparation**
4. **Modelling**
5. **Evaluation**
6. **Deployment**

Following diagram shows the different stages:



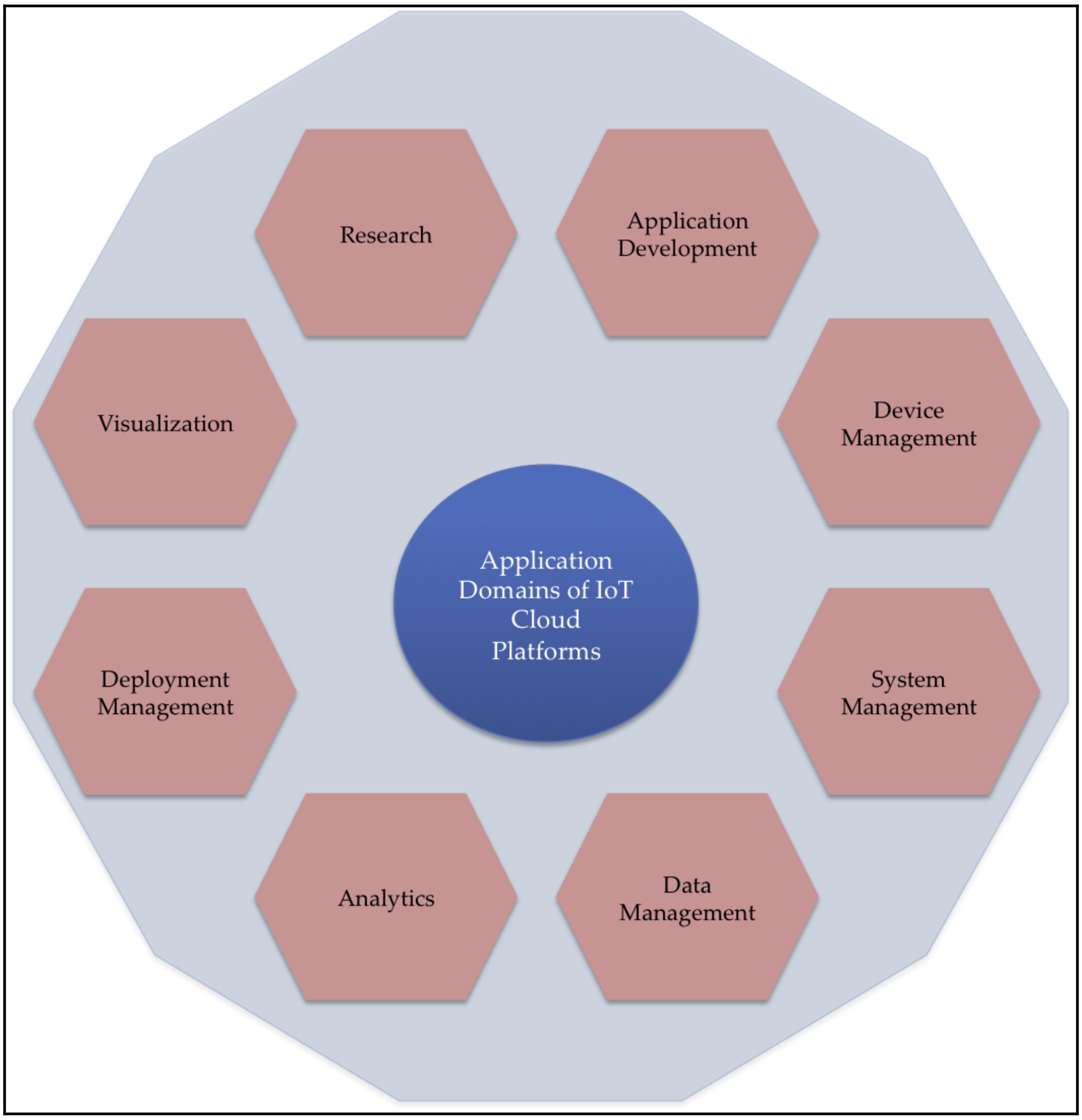
Different stages in CRISP-DM

As we can see, it's a continuous process model with data science and AI playing important roles in steps 2–5.

The details about CRISP-DM and all its phases can be read in the following:
Marbán, Óscar, Gonzalo Mariscal, and Javier Segovia. *A data mining & knowledge discovery process model*. *Data Mining and Knowledge Discovery in Real Life Applications*. InTech, 2009.

## AI platforms and IoT platforms

A large number of cloud platforms with both AI and IoT capabilities are available today. These platforms provide the capability to integrate the sensors and devices and perform analytics on the cloud. There exist more than 30 cloud platforms in the global market, each targeting different IoT verticals and services. The following screenshot lists the various services that AI/IoT platforms support:



Services that different AI/IoT platforms support

Let's briefly find out about some popular cloud platforms. In `Chapter 9`, *Personal and Home IoT*, we'll learn how to use the most popular ones. Following is a list of some of the popular Cloud platforms:

- **IBM Watson IoT Platform**: Hosted by IBM, the platform provides device management; it uses MQTT protocol to connect with IoT devices and applications. It provides real-time scalable connectivity. The data can be stored for a period and accessed in real time. IBM Watson also provides Bluemix **Platform-as-a-Service** (**PaaS**) for analytics and visualizations. We can write code to build and manage applications that interact with the data and connected devices. It supports Python along with C#, Java, and Node.js.

- **Microsoft IoT-Azure IoT suite**: It provides a collection of preconfigured solutions built on Azure PaaS. It enables a reliable and secure bidirectional communication between IoT devices and cloud. The preconfigured solutions include data visualization, remote monitoring, and configuring rules and alarms over live IoT telemetry. It also provides Azure Stream Analytics to process the data in real time. The Azure Stream Analytics allows us to use Visual Studio. It supports Python, Node.js, C, and Arduino, depending upon the IoT devices.
- **Google Cloud IoT**: The Google Cloud IoT provides a fully managed service for securely connecting and managing IoT devices. It supports both MQTT and HTTP protocols. It also provides bidirectional communication between IoT devices and the cloud. It provides support for Go, PHP, Ruby, JS, .NET, Java, Objective-C, and Python. It also has BigQuery, which allows users to perform data analytics and visualization.
- **Amazon AWS IoT**: The Amazon AWS IoT allows IoT devices to communicate via MQTT, HTTP, and WebSockets. It provides secure, bi-directional communication between IoT devices and the cloud. It also has a rules engine that can be used to integrate data with other AWS services and transform the data. Rules can be defined that trigger the execution of user code in Java, Python, or Node.js. AWS Lambda allows us to use our own custom trained models.

# Tools used in this book

For the implementation of IoT-based services, we need to follow a bottom-up approach. For each IoT vertical, we need to find the analytics and the data and, finally, implement it in code.

Due to its availability in almost all AI and IoT platforms, Python will be used for coding in this book. Along with Python, some helping libraries such as NumPy, pandas, SciPy, Keras, and TensorFlow will be used to perform AI/ML analytics on the data. For visualization, we would be using Matplotlib and Seaborn.

## TensorFlow

TensorFlow is an open source software library developed by the Google Brain team; it has functions and APIs for implementing deep neural networks. It works with Python, C++, Java, R, and Go. It can be used to work on multiple platforms, CPU, GPU, mobile, and even distributed. TensorFlow allows for model deployment and ease of use in production. The optimizer in TensorFlow makes the task of training deep neural networks easier by automatically calculating gradients and applying them to update weights and biases.

In TensorFlow, a program has two distinct components:

- **Computation graph** is a network of nodes and edges. Here all of the data, variables, placeholders, and the computations to be performed are defined. TensorFlow supports three types of data objects: constants, variables, and placeholders.
- **Execution graph** actually computes the network using a `Session` object. Actual calculations and transfer of information from one layer to another takes place in the `Session` object.

Let's see the code to perform matrix multiplication in TensorFlow. The whole code can be accessed from the GitHub repository (`https://github.com/PacktPublishing/Hands-On-Artificial-Intelligence-for-IoT`) filename, `matrix_multiplication.ipynb`:

```
import tensorflow as tf
import numpy as np
```

This part imports the TensorFlow module. Next, we define the computation graph. `mat1` and `mat2` are two matrices we need to multiply:

```
# A random matrix of size [3,5]
mat1 = np.random.rand(3,5)
# A random matrix of size [5,2]
mat2 = np.random.rand(5,2)
```

We declare two placeholders, `A` and `B`, so that we can pass their values at runtime. In the computation graph, we declare all of the data and computation objects:

```
# Declare placeholders for the two matrices
A = tf.placeholder(tf.float32, None, name='A')
B = tf.placeholder(tf.float32, None, name='B')
```

This declares two placeholders with the names `A` and `B`; the arguments to the `tf.placeholder` method specify that the placeholders are of the `float32` datatype. Since the shape specified is `None`, we can feed it a tensor of any shape and an optional name for the operation. Next, we define the operation to be performed using the matrix multiplication method, `tf.matmul`:

```
C = tf.matmul(A,B)
```

The execution graph is declared as a `Session` object, which is fed the two matrices, `mat1` and `mat2`, for the placeholders, `A` and `B`, respectively:

```
with tf.Session() as sess:
    result = sess.run(C, feed_dict={A: mat1, B:mat2})
    print(result)
```

# Keras

Keras is a high-level API that runs on top of TensorFlow. It allows for fast and easy prototyping. It supports both convolutional and recurrent neural networks and even a combination of the two. It can run on both CPU and GPU. The following code performs matrix multiplication using Keras:

```
# Import the libraries
import keras.backend as K
import numpy as np

# Declare the data
A = np.random.rand(20,500)
B = np.random.rand(500,3000)

#Create Variable
x = K.variable(value=A)
y = K.variable(value=B)
z = K.dot(x,y)
print(K.eval(z))
```

# Datasets

In the coming chapters, we'll be learning different DL models and ML methods. They all work on data; while a large number of datasets are available to demonstrate how these models work, in this book, we'll use datasets available freely through wireless sensors and other IoT devices. Following are some of the datasets used in this book and their sources.

## Combined cycle power plant

This dataset contains 9,568 data points collected from a **combined cycle power plant** (**CCPP**) in a course of six years (2006-2011). CCPP uses two turbines to generate power, the gas turbine and the steam turbine. There're three main components of the CCPP plant: gas turbine, heat recovery system, and steam turbine. The dataset available at UCI ML (`http://archive.ics.uci.edu/ml/datasets/combined+cycle+power+plant`) was collected by Pinar Tufekci from Namik Kemal University and Heysem Kaya from Bogazici University. The data consists of four features determining the average ambient variables. The averages are taken from various sensors located around the plant that record ambient variables per second. The aim is to predict the net hourly electrical energy output. The data is available in both `xls` and `ods` format.

The features in the dataset are as follows:

- The **Ambient Temperature** (**AT**) is in the range 1.81°C and 37.11°C
- The **Ambient Pressure** (**AP**) is in the range 992.89—1033.30 millibar
- **Relative Humidity** (**RH**) is in the range 25.56% to 100.16%
- Exhaust **Vacuum** (**V**) is in the range 25.36 to 81.56 cm Hg
- Net hourly electrical energy output (PE) is in the range 420.26 to 495.76 MW

Further details about the data and the problem can be read from the following:

- Pınar Tüfekci, *Prediction of full load electrical power output of a baseload operated combined cycle power plant using machine learning methods*, International Journal of Electrical Power & Energy Systems, Volume 60, September 2014, Pages 126-140, ISSN 0142-0615.
- Heysem Kaya, Pınar Tüfekci, Sadık Fikret Gürgen: *Local and GlobalLearning Methods for Predicting Power of a Combined Gas & Steam Turbine*, Proceedings of the International Conference on Emerging Trends in Computer and Electronics Engineering ICETCEE 2012, pp. 13-18 (Mar. 2012, Dubai).

## Wine quality dataset

Wineries around the world have to undergo wine certifications and quality assessments to safeguard human health. The wine certification is performed with the help of physicochemical analysis and sensory tests. With the advancement of technology, the physicochemical analysis can be performed routinely via in-vitro equipment.

We use this dataset for classification examples in this book. The dataset can be downloaded from the UCI-ML repository (`https://archive.ics.uci.edu/ml/datasets/Wine+Quality`). The wine quality dataset contains results of physicochemical tests on different samples of red and white wine. Each sample was further rated by an expert wine taster for quality on a scale of 0—10.

The dataset contains in total 4,898 instances; it has in total 12 attributes. The 12 attributes are as follows:

- Fixed acidity
- Volatile acidity
- Citric acid
- Residual sugar
- Chlorides
- Free sulfur dioxide
- Total sulfur dioxide
- Density
- pH
- Sulfates
- Alcohol
- Quality

The dataset is available in the `csv` format.

Details about the dataset can be read from this paper: Cortez, Paulo, et al. *Modeling wine preferences by data mining from physicochemical properties.* Decision Support Systems 47.4 (2009): 547-553 (`https://repositorium.sdum.uminho.pt/bitstream/1822/10029/1/wine5.pdf`).

## Air quality data

Air pollution poses a major environmental risk to human health. It's found that there exists a correlation between improved air quality and amelioration of different health problems such as respiratory infections, cardiovascular diseases, and lung cancer. The extensive sensor networks throughout the world by Meteorological Organizations of the respective country provide us with real-time air quality data. This data can be accessed through the respective web APIs of these organizations.

In this book, we'll use the historical air quality data to train our network and predict the mortality rate. The historical data for England is available freely at Kaggle (`https://www.kaggle.com/c/predict-impact-of-air-quality-on-death-rates`), and the air quality data consists of daily means of **ozone** (**O3**), **Nitrogen dioxide** (**NO2**), particulate matter with a diameter less than or equal to 10 micrometers (PM10) and PM25 (2.5 micrometers or less), and temperature. The mortality rate (number of deaths per 100,000 people) for England region is obtained by the data provided by the UK Office for National Statistics.

# Summary

In this chapter, we learned about IoT, big data, and AI. This chapter introduced the common terminologies used in IoT. We learned about the IoT architecture for data management and data analysis. The enormous data generated by IoT devices necessitates special ways to handle it.

We learned about how data science and AI can help in both analytics and prediction generated by the many IoT devices. Various IoT platforms were briefly described in this chapter, as were some popular IoT verticals. We also learned about special DL libraries: TensorFlow and Keras. Finally, some of the datasets we'll be using throughout the book were introduced.

The next chapter will cover how to access the datasets available in varied formats.

# 2
# Data Access and Distributed Processing for IoT

Data is everywhere: images, speech, text, weather information, the speed of your car, your last EMI, changing stock prices. With the integration of **Internet of Things** (**IoT**) systems, the amount of data produced has increased many fold; an example is sensor readings, which could be taken for room temperature, soil alkalinity, and more. This data is stored and made available in various formats. In this chapter, we will learn how to read, save, and process data in some popular formats. Specifically, you will do the following:

- Access data in TXT format
- Read and write CSV-formatted data via the csv, pandas, and numpy modules
- Access JSON data using JSON and pandas
- Learn to work with the HDF5 format using PyTables, pandas, and h5py
- Handle SQL databases using SQLite and MySQL
- Handle NoSQL using MongoDB
- Work with Hadoop's Distributed File System

## TXT format

One of the simplest and common formats for storing data is the TXT format; many IoT sensors log sensor readings with different timestamps in the simple `.txt` file format. Python provides inbuilt functions for creating, reading, and writing into TXT files.

We can access TXT files in Python itself without using any module; the data, in this case, is of the string type, and you will need to transform it to other types to use it. Alternatively, we can use NumPy or pandas.

# Using TXT files in Python

Python has inbuilt functions that read and write into TXT files. The complete functionality is provided using four sets of functions: `open()`, `read()`, `write()`, and `close()`. As the names suggest, they are used to open a file, read from a file, write into a file, and finally close it. If you are dealing with string data (text), this is the best choice. In this section, we will use `Shakespeare` plays in TXT form; the file can be downloaded from the MIT site: `https://ocw.mit.edu/ans7870/6/6.006/s08/lecturenotes/files/t8.shakespeare.txt`.

We define the following variables to access the data:

```
data_folder = '../../data/Shakespeare'
data_file = 'alllines.txt'
```

The first step here is to open the file:

```
f = open(data_file)
```

Next, we read the whole file; we can use the `read` function, which will read the whole file as one single string:

```
contents = f.read()
```

This reads the whole file (consisting of 4,583,798 characters) into the `contents` variable. Let's explore the contents of the `contents` variable, which will print the first `1000` characters:

```
print(contents[:1000])
```

The preceding code will print the output as follows:

```
"ACT I"
"SCENE I. London. The palace."
"Enter KING HENRY, LORD JOHN OF LANCASTER, the EARL of WESTMORELAND, SIR
WALTER BLUNT, and others"
"So shaken as we are, so wan with care,"
"Find we a time for frighted peace to pant,"
"And breathe short-winded accents of new broils"
"To be commenced in strands afar remote."
"No more the thirsty entrance of this soil"
"will daub her lips with her own children's blood,"
"Nor more will trenching war channel her fields,"
"Nor bruise her flowerets with the armed hoofs"
"Of hostile paces: those opposed eyes,"
"Which, like the meteors of a troubled heaven,"
"All of one nature, of one substance bred,"
```

```
"Did lately meet in the intestine shock"
"And furious close of civil butchery"
"will now, in mutual well-beseeming ranks,"
"March all one way and be no more opposed"
"Against acquaintance, kindred and allies:"
"The edge of war, like an ill-sheathed knife,"
"No more will cut his master. Therefore, friends,"
"As far as to the sepulchre of Christ,"
"Whose
```

If the TXT files contain numeric data, it is better to use NumPy; if data is mixed, pandas is the best choice.

# CSV format

**Comma-separated value** (**CSV**) files are the most popular formats for storing tabular data generated by IoT systems. In a `.csv` file, the values of the records are stored in plain-text rows, with each row containing the values of the fields separated by a separator. The separator is a comma by default but can be configured to be any other character. In this section, we will learn how to use data from CSV files with Python's `csv`, `numpy`, and `pandas` modules. We will use the `household_power_consumption` data file. The file can be downloaded from the following GitHub link: `https://github.com/ahanse/machlearning/blob/master/household_power_consumption.csv`. To access the data files, we define the following variables:

```
data_folder = '../../data/household_power_consumption'
data_file = 'household_power_consumption.csv'
```

Generally, to quickly read the data from CSV files, use the Python `csv` module; however, if the data needs to be interpreted as a mix of date, text, and numeric data fields, it's better use the pandas package. If the data is only numeric, NumPy is the most appropriate package.

# Working with CSV files with the csv module

In Python, the `csv` module provides classes and methods for reading and writing CSV files. The `csv.reader` method creates a reader object from which rows can be read iteratively. Each time a row is read from the file, the reader object returns a list of fields. For example, the following code demonstrates reading the data file and printing rows:

```
import csv
import os
```

```
with open(os.path.join(data_folder,data_file),newline='') as csvfile:
    csvreader = csv.reader(csvfile)
    for row in csvreader:
        print(row)
```

The rows are printed as a list of field values:

```
['date', 'time', 'global_active_power', 'global_reactive_power', 'voltage',
'global_intensity', 'sub_metering_1', 'sub_metering_2', 'sub_metering_3']
['0007-01-01', '00:00:00', '2.58', '0.136', '241.97', '10.6', '0', '0',
'0'] ['0007-01-01', '00:01:00', '2.552', '0.1', '241.75', '10.4', '0', '0',
'0'] ['0007-01-01', '00:02:00', '2.55', '0.1', '241.64', '10.4', '0', '0',
'0']
```

The `csv.writer` method returns an object that can be used to write rows to a file. As an example, the following code writes the first 10 rows of the file to a temporary file and then prints it:

```
# read the file and write first ten rows
with open(os.path.join(data_folder, data_file), newline='') as csvfile, \
        open(os.path.join(data_folder, 'temp.csv'), 'w', newline='') as
tempfile:
    csvreader = csv.reader(csvfile)
    csvwriter = csv.writer(tempfile)
    for row, i in zip(csvreader, range(10)):
        csvwriter.writerow(row)
# read and print the newly written file
with open(os.path.join(data_folder, 'temp.csv'), newline='') as tempfile:
    csvreader = csv.reader(tempfile)
    for row in csvreader:
        print(row)
```

The `delimiter` field and the `quoting` field characters are important attributes that you can set while creating `reader` and `writer` objects.

By default, the `delimiter` field is `,` and the other delimiters are specified with the `delimiter` argument to the `reader` or `writer` functions. For example, the following code saves the file with `|` as `delimiter`:

```
    # read the file and write first ten rows with '|' delimiter
with open(os.path.join(data_folder, data_file), newline='') as csvfile, \
        open(os.path.join(data_folder, 'temp.csv'), 'w', newline='') as
tempfile:
    csvreader = csv.reader(csvfile)
    csvwriter = csv.writer(tempfile, delimiter='|')
    for row, i in zip(csvreader, range(10)):
        csvwriter.writerow(row)
```

```
# read and print the newly written file
with open(os.path.join(data_folder, 'temp.csv'), newline='') as tempfile:
    csvreader = csv.reader(tempfile, delimiter='|')
    for row in csvreader:
        print(row)
```

If you do not specify a `delimiter` character when the file is read, the rows will be read as one field and printed as follows:

```
['0007-01-01|00:00:00|2.58|0.136|241.97|10.6|0|0|0']
```

`quotechar` specifies a character with which to surround fields. The `quoting` argument specifies what kind of fields can be surrounded with `quotechar`. The `quoting` argument can have one of the following values:

- `csv.QUOTE_ALL`: All the fields are quoted
- `csv.QUOTE_MINIMAL`: Only fields containing special characters are quoted
- `csv.QUOTE_NONNUMERIC`: All non-numeric fields are quoted
- `csv.QUOTE_NONE`: None of the fields are quoted

As an example, let's print the temp file first:

```
0007-01-01|00:00:00|2.58|0.136|241.97|10.6|0|0|0
0007-01-01|00:01:00|2.552|0.1|241.75|10.4|0|0|0
0007-01-01|00:02:00|2.55|0.1|241.64|10.4|0|0|0
0007-01-01|00:03:00|2.55|0.1|241.71|10.4|0|0|0
0007-01-01|00:04:00|2.554|0.1|241.98|10.4|0|0|0
0007-01-01|00:05:00|2.55|0.1|241.83|10.4|0|0|0
0007-01-01|00:06:00|2.534|0.096|241.07|10.4|0|0|0
0007-01-01|00:07:00|2.484|0|241.29|10.2|0|0|0
0007-01-01|00:08:00|2.468|0|241.23|10.2|0|0|0
```

Now let's save it with all fields quoted:

```
# read the file and write first ten rows with '|' delimiter, all quoting
and * as a quote charachetr.
with open(os.path.join(data_folder, data_file), newline='') as csvfile, \
        open('temp.csv', 'w', newline='') as tempfile:
    csvreader = csv.reader(csvfile)
    csvwriter = csv.writer(tempfile, delimiter='|',
quotechar='*',quoting=csv.QUOTE_ALL)
    for row, i in zip(csvreader, range(10)):
        csvwriter.writerow(row)
```

The file gets saved with the specified quote character:

```
*0007-01-01*|*00:00:00*|*2.58*|*0.136*|*241.97*|*10.6*|*0*|*0*|*0*
```

```
*0007-01-01*|*00:01:00*|*2.552*|*0.1*|*241.75*|*10.4*|*0*|*0*|*0*
*0007-01-01*|*00:02:00*|*2.55*|*0.1*|*241.64*|*10.4*|*0*|*0*|*0*
*0007-01-01*|*00:03:00*|*2.55*|*0.1*|*241.71*|*10.4*|*0*|*0*|*0*
*0007-01-01*|*00:04:00*|*2.554*|*0.1*|*241.98*|*10.4*|*0*|*0*|*0*
*0007-01-01*|*00:05:00*|*2.55*|*0.1*|*241.83*|*10.4*|*0*|*0*|*0*
*0007-01-01*|*00:06:00*|*2.534*|*0.096*|*241.07*|*10.4*|*0*|*0*|*0*
*0007-01-01*|*00:07:00*|*2.484*|*0*|*241.29*|*10.2*|*0*|*0*|*0*
*0007-01-01*|*00:08:00*|*2.468*|*0*|*241.23*|*10.2*|*0*|*0*|*0*
```

Remember to read the file with the same arguments; otherwise, the `*` quote character will be treated as part of the field values and printed as follows:

```
['*0007-01-01*', '*00:00:00*', '*2.58*', '*0.136*', '*241.97*', '*10.6*',
'*0*', '*0*', '*0*']
```

Using the correct arguments with the `reader` object prints the following:

```
['0007-01-01', '00:00:00', '2.58', '0.136', '241.97', '10.6', '0', '0',
'0']
```

Now let's see how we can read CSV files with pandas, another popular Python library.

# Working with CSV files with the pandas module

In pandas, the `read_csv()` function returns a DataFrame after reading the CSV file:

```
df = pd.read_csv('temp.csv')
print(df)
```

The DataFrame is printed as follows:

```
          date      time  global_active_power  global_reactive_power
voltage  \
0  0007-01-01  00:00:00                2.580                  0.136
241.97
1  0007-01-01  00:01:00                2.552                  0.100
241.75
2  0007-01-01  00:02:00                2.550                  0.100
241.64
3  0007-01-01  00:03:00                2.550                  0.100
241.71
4  0007-01-01  00:04:00                2.554                  0.100
241.98
5  0007-01-01  00:05:00                2.550                  0.100
241.83
6  0007-01-01  00:06:00                2.534                  0.096
241.07
```

```
7  0007-01-01  00:07:00               2.484                0.000
241.29
8  0007-01-01  00:08:00               2.468                0.000
241.23

   global_intensity  sub_metering_1  sub_metering_2  sub_metering_3
0              10.6               0               0               0
1              10.4               0               0               0
2              10.4               0               0               0
3              10.4               0               0               0
4              10.4               0               0               0
5              10.4               0               0               0
6              10.4               0               0               0
7              10.2               0               0               0
8              10.2               0               0               0
```

We see in the preceding output that pandas automatically interpreted the `date` and `time` columns as their respective data types. The pandas DataFrame can be saved to a CSV file with the `to_csv()` function:

```
df.to_csv('temp1.cvs')
```

pandas, when it comes to reading and writing CSV files, offers plenty of arguments. Some of these are as follows, complete with how they're used:

- `header`: Defines the row number to be used as a header, or none if the file does not contain any headers.
- `sep`: Defines the character that separates fields in rows. By default, the value of `sep` is set to `,`.
- `names`: Defines column names for each column in the file.
- `usecols`: Defines columns that need to be extracted from the CSV file. Columns that are not mentioned in this argument are not read.
- `dtype`: Defines the data types for columns in the DataFrame.

Many other available options are documented at the following links: `https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html` and `https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_csv.html`.

Now let's see how to read data from CSV files with the NumPy module.

## Working with CSV files with the NumPy module

The NumPy module provides two functions for reading values from CSV files: `np.loadtxt()` and `np.genfromtxt()`.

An example of `np.loadtxt` is as follows:

```
arr = np.loadtxt('temp.csv', skiprows=1, usecols=(2,3), delimiter=',')
arr
```

The preceding code reads columns `3` and `4` from the file that we created earlier, and saves them in a 9 × 2 array as follows:

```
array([[2.58 , 0.136],
       [2.552, 0.1  ],
       [2.55 , 0.1  ],
       [2.55 , 0.1  ],
       [2.554, 0.1  ],
       [2.55 , 0.1  ],
       [2.534, 0.096],
       [2.484, 0.   ],
       [2.468, 0.   ]])
```

The `np.loadtxt()` function cannot handle CSV files with missing data. For instances where data is missing, `np.genfromtxt()` can be used. Both of these functions offer many more arguments; details can be found in the NumPy documentation. The preceding code can be written using `np.genfromtxt()` as follows:

```
arr = np.genfromtxt('temp.csv', skip_header=1, usecols=(2,3),
delimiter=',')
```

NumPy arrays produced as a result of applying AI to IoT data can be saved with `np.savetxt()`. For example, the array we loaded previously can be saved as follows:

```
np.savetxt('temp.csv', arr, delimiter=',')
```

The `np.savetxt()` function also accepts various other useful arguments, such as the format for saved fields and headers. Check the NumPy documentation for more details on this function.

CSV is the most popular data format on IoT platforms and devices. In this section, we learned how to read CSV data using three different packages in Python. Let's learn about JSON, another popular format, in the next section.

# XLSX format

Excel, a component of the Microsoft Office pack, is one of the popular formats in which data is stored and visualized. Since 2010, Office has supported the `.xlsx` format. We can read XLSX files using the OpenPyXl and pandas functions.

# Using OpenPyXl for XLSX files

OpenPyXl is a Python library for reading and writing Excel files. It is an open-source project. A new `workbook` is created using the following command:

```
wb = Workbook()
```

We can access the currently `active` sheet by using the following command:

```
ws = wb.active()
```

To change the sheet name, use the `title` command:

```
ws.title = "Demo Name"
```

A single row can be added to the sheet using the `append` method:

```
ws.append()
```

A new sheet can be created using the `create_sheet()` method. An individual cell in the active sheet can be created using the `column` and `row` values:

```
# Assigns the cell corresponding to
# column A and row 10 a value of 5
ws.['A10'] = 5
#or
ws.cell(column=1, row=10, value=5)
```

A workbook can be saved using the `save` method. To load an existing workbook, we can use the `load_workbook` method. The names of the different sheets in an Excel workbook can be accessed using `get_sheet_names()`.

The following code creates an Excel workbook with three sheets and saves it; later it loads the sheet and accesses a cell. The code can be accessed from GitHub at `OpenPyXl_example.ipynb`:

```
# Creating and writing into xlsx file
from openpyxl import Workbook
from openpyxl.compat import range
from openpyxl.utils import get_column_letter
wb = Workbook()
dest_filename = 'empty_book.xlsx'
ws1 = wb.active
ws1.title = "range names"
for row in range(1, 40):
 ws1.append(range(0,100,5))
ws2 = wb.create_sheet(title="Pi")
ws2['F5'] = 2 * 3.14
ws2.cell(column=1, row=5, value= 3.14)
ws3 = wb.create_sheet(title="Data")
for row in range(1, 20):
 for col in range(1, 15):
 _ = ws3.cell(column=col, row=row, value="\
 {0}".format(get_column_letter(col)))
print(ws3['A10'].value)
wb.save(filename = dest_filename)

# Reading from xlsx file
from openpyxl import load_workbook
wb = load_workbook(filename = 'empty_book.xlsx')
sheet_ranges = wb['range names']
print(wb.get_sheet_names())
print(sheet_ranges['D18'].value)
```

You can learn more about OpenPyXL from its documentation, available at `https://openpyxl.readthedocs.io/en/stable/`.

## Using pandas with XLSX files

We can load existing `.xlsx` files with the help of pandas. The `read_excel` method is used to read Excel files as a DataFrame. This method uses an argument, `sheet_name`, which is used to specify the sheet we want to load. The sheet name can be specified either as a string or number starting from zero. The `to_excel` method can be used to write into an Excel file.

The following code reads an Excel file, manipulates it, and saves it. The code can be accessed from GitHub at `Pandas_xlsx_example.ipynb`:

```
import pandas as pd
df = pd.read_excel("empty_book.xlsx", sheet_name=0)
df.describe()
result = df * 2
result.describe()
result.to_excel("empty_book_modified.xlsx")
```

# Working with the JSON format

**JavaScript Object Notation** (**JSON**) is another popular data format in IoT systems. In this section, we will learn how to read JSON data with Python's JSON, NumPy, and pandas packages.

For this section, we will use the `zips.json` file, which contains US ZIP codes with city codes, geolocation details, and state codes. The file has JSON objects recorded in the following format:

```
{ "_id" : "01001", "city" : "AGAWAM", "loc" : [ -72.622739, 42.070206 ],
"pop" : 15338, "state" : "MA" }
```

# Using JSON files with the JSON module

To load and decode JSON data, use the `json.load()` or `json.loads()` functions. As an example, the following code reads the first 10 lines from the `zips.json` file and prints them nicely:

```
import os
import json
from pprint import pprint

with open(os.path.join(data_folder,data_file)) as json_file:
    for line,i in zip(json_file,range(10)):
        json_data = json.loads(line)
        pprint(json_data)
```

The objects are printed as follows:

```
{'_id': '01001',
 'city': 'AGAWAM',
 'loc': [-72.622739, 42.070206],
 'pop': 15338,
 'state': 'MA'}
```

The `json.loads()` function takes string objects as input while the `json.load()` function takes file objects as input. Both functions decode the JSON object and load it in the `json_data` file as a Python dictionary object.

The `json.dumps()` function takes an object and produces a JSON string, and the