**1.INTRODUCTION**

Nowadays, people are paying more attention to their health, but there are still a lot of dangerous behaviors that may get people injured. They are extremely threatening in some specific situations. For example, talking on the phone while driving distracts people's attention, which may result in traffic accidents. Also, smoking is prohibited in places such as gas stations and department stores, since they may cause fires or even explosions. Avoiding some bad behaviors may save many people's lives and therefore governments have already implemented lots of regulation on people's behavior improperly and they need to be detected in time. However, it is impossible to detect all these behaviors simply by human beings. Fortunately, Machine learning and Computer vision is becoming more prevailing and can be used by humans. So, if some smoking and calling images can be put into computers for learning, they can be used to help detect improper behaviors.

Machine learning developed significantly in different fields in recent years. In the previous studies, there are some studies that have already tried to apply machine learning into the field of computer vision about humans. What's more, Zhu et al. also gave out an algorithm based on deep learning to monitor students' behaviors during the test. In terms of smoking behavior detection, Zhang et al. have developed a machine learning algorithm in the method of decision tree [6]. Their model achieved 84.11% accuracy with the best performance.

However, there are still few studies about the prediction of calling behaviors, especially applying algorithms based on Machine Learning methods. For instance, smoking, talking on the phone is hard to detect even with our naked eyes. The phone may be too small to be blocked by people's hands, thus making the problem more complicated. In [7], Zheng used Machine Learning algorithms based on Support Vector Machine (SVM) as well as Convolutional Neural Network (CNN) to predict people's walking upstairs and downstairs behavior, which achieved 93.5% as the highest accuracy. However, this paper would like to compare the mainstream machine learning algorithms in detecting the smoking and calling behaviors and figure out which one is the best solution to the problem.

**2.LITERATURE SURVEY**

# **1. Research on Recommendation of Insurance Products Based on Random Forest**

**Authors :** [**Yan Guo**](https://ieeexplore.ieee.org/author/37087225605)**;** [**Yu Zhou**](https://ieeexplore.ieee.org/author/37089113549)**;** [**Xiaonan Hu**](https://ieeexplore.ieee.org/author/37087227178)**;** [**Wenchuan Cheng**](https://ieeexplore.ieee.org/author/37087325116)

With the rapid development of recommendation systems, how to predict user's behavior accurately becomes more and more important. In this paper, random forest is applied to recommend insurance products and compared with ID3, C4.5, Naive-Bayes and Nearest-neighbor. Experiment results show that the prediction error of random forest is 2.02% lower than ID3, 1.09% lower than C4.5, 1.67% lower than Naive-Bayes and 5.97% lower than Nearest-neighbor.

**2. Clustering Analysis for Silent Telecom Customers Based on K-means++**

**Authors : Y. Qiu, P. Chen, Z. Lin, Y. Yang, L. Zeng and Y. Fan**

Silent customers are part of customers that company is very easy to lose. It is necessary to analyze the features of such customers and make appropriate market decisions to improve the enterprise's revenue in the telecom industry. This paper proposes a K-means++ method for customer segmentation based on silent customers. Firstly, key variables to the segmentation model were screened out and then the original data was preprocessed. Secondly, silent customers were clustered and the Calinski-Harabasz index was adopted to verify the best clustering effect when k=6.At last, radar chart analysis and suggestions were given, which would provide data supports to the improvement of operation and maintenance management and decision-making of the precision marketing.

# **3. Research on Pedestrian Attitude Detection Algorithm from the Perspective of Machine Learning**

**Authors : Kailun Wan**

In the rapid development of science and technology today, the intelligence of the visual system has been highly valued. The recognition and detection of pedestrian attitudes in a complex environment have become the application trend of intelligent video. The widely used camera machine does not have such a function. Therefore, this article deeply discusses the relevant algorithms of pedestrian gesture detection and recognition based on machine learning. The traditional HOG feature detection can only achieve the relevant detection of the upright walking crowd target. While when the pedestrian makes different gestures, its detection effect is directly affected and challenging to be recognized. So, this article uses the checking methods of the deformable part model (DPM) to check the target pedestrian gesture and elaborate pedestrian's attitude estimation algorithm for the deformable parts principle

**4.** **Prediction of Daily Smoking Behavior Based on Decision Tree Machine Learning Algorithm**

**Authors : Y. Zhang, J. Liu, Z. Zhang and J. Huang**

With the accumulation of smoking data and the development of the algorithm, precise analysis becomes possible and this can benefit smoking cessation a lot. However, as far as we know, little research has been done on the behavior of everyday smoking, such as the precise time when a smoker smokes. This paper proposes a model based on a decision tree machine learning algorithm to predict daily smoking time. The simulation data set of smoking time data was established by using the population information of smokers collected by the Chinese center for disease control and prevention. In order to solve the problem of too little feature information, we propose a feature information extraction module

**5. Video-based Human Action Recognition using Deep Learning**

**Authors : Hieu H. Pham, Louahdi Khoudour, Alain Crouzil, Pablo Zegers, and Sergio A. Velastin**

Human action recognition is an important application domain in computer vision. Its primary aim is to accurately describe human actions and their interactions from a previously unseen data sequence acquired by sensors. The ability to recognize, understand and predict complex human actions enables the construction of many important applications such as intelligent surveillance systems, human-computer interfaces, health care, security and military applications. In recent years, deep learning has been given particular attention by the computer vision community. This paper presents an overview of the current state-of-the-art in action recognition using video analysis with deep learning techniques. We present the most important deep learning models for recognizing human actions, analyze them to provide the current progress of deep learning algorithms applied to solve human action recognition problems in realistic videos highlighting their advantages and disadvantages.

.

**3.SYSTEM SPECIFICATION:**

**HARDWARE REQUIREMENTS:**

* **System :** Intel i3
* **Hard Disk :** 1 TB.
* **Monitor** : 14’ Colour Monitor.
* **Mouse :** Optical Mouse.
* **Ram :** 4GB.

**SOFTWARE REQUIREMENTS:**

* **Operating system :** Windows 10.
* **Coding Language :** Python.
* **Front-End :** HTML. CSS
* **Designing :** HTML ,CSS ,JavaScript.
* **Data Base :** SQLite.

# **4.SYSTEM ANALYSIS**

**EXISTING SYSTEM:**

In the previous studies, there are some studies that have already tried to apply machine learning into the field of computer vision about humans. By designing a convolutional neural network, the computer managed to distinguish different human's behaviors. What's more, Zhu et al. also gave out an algorithm based on deep learning to monitor students' behaviors during the test. In terms of smoking behavior detecting, Zhang et al. have developed a machine learning algorithm in the method of decision tree. Their model achieved 84.11% accuracy with the best performance

**DISADVANTAGES OF EXISTING SYSTEM:**

1. Results are not up to the mark.
2. The dataset this paper uses only one class(type).

**Algorithm:** **SVM, Decision tree, k-means**

**PROPOSED SYSTEM:**

For instance, smoking, talking on the phone is hard to detect even with our naked eyes as well. The phone may be too small to be blocked by people's hands, thus making the problem more complicated. Zheng used Machine Learning algorithms based on Support Vector Machine (SVM) as well as Convolutional Neural Network (CNN) to predict people's walking upstairs and downstairs behavior, which achieved 93.5% as the highest accuracy. However, this paper would like to compare the mainstream machine learning algorithms in detecting the smoking and calling behaviors and figure out which one is the best solution to the problem. The rest of this paper is divided into following parts: Part 2 shows the datasets this study picked and its problems to be solved. Then visualization of each class and results of different machine learning algorithms will be presented in Part

3. Finally, the conclusion will be summarized in Part 4.

**ADVANTAGES OF PROPOSED SYSTEM:**

* achieved 93.5% as the highest accuracy.
* The dataset this paper uses has three classes: Smoking class, Calling class and Normal class.

**Algorithm:** Convolutional Neural Network (CNN), Machine Learning algorithms, Support Vector Machine (SVM)

**5.SYSTEM STUDY**

**5.1 FEASIBILITY STUDY**

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

**Three key considerations involved in the feasibility analysis are,**

• ECONOMICAL FEASIBILITY

• TECHNICAL FEASIBILITY

• SOCIAL FEASIBILITY

**5.2 ECONOMICAL FEASIBILITY**

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

**5.3 TECHNICAL FEASIBILITY**

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

**5.4 SOCIAL FEASIBILITY**

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

**6.SOFTWARE ENVIRONMENT**

**6.1 PYTHON**

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. An [interpreted language](https://en.wikipedia.org/wiki/Interpreted_language), Python has a design philosophy that emphasizes code [readability](https://en.wikipedia.org/wiki/Readability) (notably using [whitespace](https://en.wikipedia.org/wiki/Whitespace_character) indentation to delimit [code blocks](https://en.wikipedia.org/wiki/Code_block) rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer [lines of code](https://en.wikipedia.org/wiki/Source_lines_of_code) than might be used in languages such as [C++](https://en.wikipedia.org/wiki/C%2B%2B)or [Java](https://en.wikipedia.org/wiki/Java_(programming_language)). It provides constructs that enable clear programming on both small and large scales. Python interpreters are available for many [operating systems](https://en.wikipedia.org/wiki/Operating_system). [CPython](https://en.wikipedia.org/wiki/CPython" \o "CPython), the [reference implementation](https://en.wikipedia.org/wiki/Reference_implementation) of Python, is [open source](https://en.wikipedia.org/wiki/Open_source) software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit [Python Software Foundation](https://en.wikipedia.org/wiki/Python_Software_Foundation). Python features a [dynamic type](https://en.wikipedia.org/wiki/Dynamic_type) system and automatic [memory management](https://en.wikipedia.org/wiki/Memory_management). It supports multiple [programming paradigms](https://en.wikipedia.org/wiki/Programming_paradigm), including [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming), [imperative](https://en.wikipedia.org/wiki/Imperative_programming), [functional](https://en.wikipedia.org/wiki/Functional_programming) and [procedural](https://en.wikipedia.org/wiki/Procedural_programming), and has a large and comprehensive [standard library](https://en.wikipedia.org/wiki/Standard_library).

**6.1.1 Interactive Mode Programming**

Invoking the interpreter without passing a script file as a parameter brings up the following prompt −

$ python

Python 2.4.3 (#1, Nov 11 2010, 13:34:43)

[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2

Type "help", "copyright", "credits" or "license" for more information.

>>>

Type the following text at the Python prompt and press the Enter −

>>> print "Hello, Python!"

If you are running new version of Python, then you would need to use print statement with parenthesis as in print ("Hello, Python!");. However in Python version 2.4.3, this produces the following result −

Hello, Python!

**6.1.2 Script Mode Programming**

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. Python files have extension .py. Type the following source code in a test.py file −

Live Demo

print "Hello, Python!"

We assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows −

$ python test.py

This produces the following result −

Hello, Python!

Let us try another way to execute a Python script. Here is the modified test.py file −

Live Demo

#!/usr/bin/python

print "Hello, Python!"

We assume that you have Python interpreter available in /usr/bin directory. Now, try to run this program as follows −

$ chmod +x test.py # This is to make file executable

$./test.py

This produces the following result −

Hello, Python!

**6.1.3 Python Identifiers**

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (\_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in Python.

Here are naming conventions for Python identifiers −

Class names start with an uppercase letter. All other identifiers start with a lowercase letter.

Starting an identifier with a single leading underscore indicates that the identifier is private.

Starting an identifier with two leading underscores indicates a strongly private identifier.

If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

**6.1.4 Reserved Words**

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

and exec not

assert finally or

break for pass

class from print

continue global raise

def if return

del import try

elif in while

else is with

except lambda yield

**6.1.5 Lines and Indentation**

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example −

if True:

print "True"

else:

print "False"

However, the following block generates an error −

if True:

print "Answer"

print "True"

else:

print "Answer"

print "False"

Thus, in Python all the continuous lines indented with same number of spaces would form a block. The following example has various statement blocks −

Note − Do not try to understand the logic at this point of time. Just make sure you understood various blocks even if they are without braces.

#!/usr/bin/python

import sys

try:

# open file stream

file = open(file\_name, "w")

except IOError:

print "There was an error writing to", file\_name

sys.exit()

print "Enter '", file\_finish,

print "' When finished"

while file\_text != file\_finish:

file\_text = raw\_input("Enter text: ")

if file\_text == file\_finish:

# close the file

file.close

break

file.write(file\_text)

file.write("\n")

file.close()

file\_name = raw\_input("Enter filename: ")

if len(file\_name) == 0:

print "Next time please enter something"

sys.exit()

try:

file = open(file\_name, "r")

except IOError:

print "There was an error reading file"

sys.exit()

file\_text = file.read()

file.close()

print file\_text

**6.1.6 Multi-Line Statements**

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue. For example −

total = item\_one + \

item\_two + \

item\_three

Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example −

days = ['Monday', 'Tuesday', 'Wednesday','Thursday', 'Friday']

**6.1.7 Quotation in Python**

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines. For example, all the following are legal −

word = 'word'

sentence = "This is a sentence."

paragraph = """This is a paragraph. It is

made up of multiple lines and sentences."""

**6.1.8 Comments in Python**

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

Live Demo

#!/usr/bin/python

# First comment

print "Hello, Python!" # second comment

This produces the following result −

Hello, Python!

You can type a comment on the same line after a statement or expression −

name = "Madisetti" # This is again comment

You can comment multiple lines as follows −

# This is a comment.

# This is a comment, too.

# This is a comment, too.

# I said that already.

Following triple-quoted string is also ignored by Python interpreter and can be used as a multiline comments:

'''This is a multiline comment.'''

**6.1.9 Using Blank Lines**

A line containing only whitespace, possibly with a comment, is known as a blank line and Python totally ignores it.

In an interactive interpreter session, you must enter an empty physical line to terminate a multiline statement.

**6.1.10 Waiting for the User**

The following line of the program displays the prompt, the statement saying “Press the enter key to exit”, and waits for the user to take action −

#!/usr/bin/python

raw\_input("\n\nPress the enter key to exit.")

Here, "\n\n" is used to create two new lines before displaying the actual line. Once the user presses the key, the program ends. This is a nice trick to keep a console window open until the user is done with an application.Multiple Statements on a Single Line

The semicolon ( ; ) allows multiple statements on the single line given that neither statement starts a new code block. Here is a sample snip using the semicolon.

import sys; x = 'foo'; sys.stdout.write(x + '\n')

**6.1.11 Multiple Statement Groups as Suites**

A group of individual statements, which make a single code block are called suites in Python. Compound or complex statements, such as if, while, def, and class require a header line and a suite.

Header lines begin the statement (with the keyword) and terminate with a colon ( : ) and are followed by one or more lines which make up the suite. For example −

if expression :

suite

elif expression :

suite

else :

suite

**6.1.12 Command Line Arguments**

Many programs can be run to provide you with some basic information about how they should be run. Python enables you to do this with -h −

$ python -h

usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...

Options and arguments (and corresponding environment variables):

-c cmd : program passed in as string (terminates option list)

-d : debug output from parser (also PYTHONDEBUG=x)

-E : ignore environment variables (such as PYTHONPATH)

-h : print this help message and exit

You can also program your script in such a way that it should accept various options. Command Line Arguments is an advanced topic and should be studied a bit later once you have gone through rest of the Python concepts.

**6.1.13 Python Lists**

The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets. For example −

list1 = ['physics', 'chemistry', 1997, 2000];

list2 = [1, 2, 3, 4, 5 ];

list3 = ["a", "b", "c", "d"]

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

**6.1.14 Python tuples**

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also. For example −

tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5 );

tup3 = "a", "b", "c", "d";

The empty tuple is written as two parentheses containing nothing −

tup1 = ();

To write a tuple containing a single value you have to include a comma, even though there is only one value −

tup1 = (50,);

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

**6.1.14.1 Accessing Values in Tuples**

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example −

Live Demo

#!/usr/bin/python

tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5, 6, 7 );

print "tup1[0]: ", tup1[0];

print "tup2[1:5]: ", tup2[1:5];

When the above code is executed, it produces the following result −

tup1[0]: physics

tup2[1:5]: [2, 3, 4, 5]

**6.1.14.2 Delete Tuple Elements**

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the del statement. For example −

Live Demo

#!/usr/bin/python

tup = ('physics', 'chemistry', 1997, 2000);

print tup;

del tup;

print "After deleting tup : ";

print tup;

This produces the following result. Note an exception raised, this is because after del tup tuple does not exist any more −

('physics', 'chemistry', 1997, 2000)

After deleting tup :

Traceback (most recent call last):

File "test.py", line 9, in <module>print tup;

NameError: name 'tup' is not defined

**6.1.15.1 Accessing Values in Dictionary**

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example −

Live Demo

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

print "dict['Name']: ", dict['Name']

print "dict['Age']: ", dict['Age']

When the above code is executed, it produces the following result −

Live Demo

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

print "dict['Alice']: ", dict['Alice']

When the above code is executed, it produces the following result −

dict['Alice']:

Traceback (most recent call last):

File "test.py", line 4, in <module>

print "dict['Alice']: ", dict['Alice'];

KeyError: 'Alice'

**6.1.15.2 Updating Dictionary**

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example −

Live Demo

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

dict['Age'] = 8; # update existing entry

dict['School'] = "DPS School"; # Add new entry

print "dict['Age']: ", dict['Age']

print "dict['School']: ", dict['School']

When the above code is executed, it produces the following result −

dict['Age']: 8

dict['School']: DPS School

**6.1.15.3 Delete Dictionary Elements**

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the del statement. Following is a simple example −

Live Demo

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

del dict['Name']; # remove entry with key 'Name'

dict.clear(); # remove all entries in dict

del dict ; # delete entire dictionary

print "dict['Age']: ", dict['Age']

print "dict['School']: ", dict['School']

This produces the following result. Note that an exception is raised because after del dict dictionary does not exist any more −

dict['Age']:

Traceback (most recent call last):

File "test.py", line 8, in <module>

print "dict['Age']: ", dict['Age'];

TypeError: 'type' object is unsubscriptable

Note − del() method is discussed in subsequent section.

**6.1.15.4 Properties of Dictionary Keys**

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

There are two important points to remember about dictionary keys −

(a) More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins. For example −

Live Demo

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}

print "dict['Name']: ", dict['Name']

When the above code is executed, it produces the following result −

dict['Name']: Manni

(b) Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed. Following is a simple example −

Live Demo

#!/usr/bin/python

dict = {['Name']: 'Zara', 'Age': 7}

print "dict['Name']: ", dict['Name']

When the above code is executed, it produces the following result −

Traceback (most recent call last):

File "test.py", line 3, in <module>

dict = {['Name']: 'Zara', 'Age': 7};

TypeError: unhashable type: 'list'

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates −

Live Demo

#!/usr/bin/python

tup1 = (12, 34.56);

tup2 = ('abc', 'xyz');

# Following action is not valid for tuples

# tup1[0] = 100;

# So let's create a new tuple as follows

tup3 = tup1 + tup2;

print tup3;

When the above code is executed, it produces the following result −

(12, 34.56, 'abc', 'xyz')

**6.2 DJANGO**

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source.

Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes [reusability](https://en.wikipedia.org/wiki/Reusability" \o "Reusability)and "pluggability" of components, rapid development, and the principle of [don't repeat yourself](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself). Python is used throughout, even for settings files and data models.



Django also provides an optional administrative [create, read, update and delete](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete) interface that is generated dynamically through [introspection](https://en.wikipedia.org/wiki/Introspection_(computer_science)) and configured via admin models



**6.2.1 Create a Project**

Whether you are on Windows or Linux, just get a terminal or a cmd prompt and navigate to the place you want your project to be created, then use this code −

$ django-admin startproject myproject

This will create a "myproject" folder with the following structure −

myproject/

manage.py

myproject/

\_\_init\_\_.py

settings.py

urls.py

wsgi.py

**6.2.2 The Project Structure**

The “myproject” folder is just your project container, it actually contains two elements −

manage.py − This file is kind of your project local django-admin for interacting with your project via command line (start the development server, sync db...). To get a full list of command accessible via manage.py you can use the code −

$ python manage.py help

The “myproject” subfolder − This folder is the actual python package of your project. It contains four files −

\_\_init\_\_.py − Just for python, treat this folder as package.

settings.py − As the name indicates, your project settings.

urls.py − All links of your project and the function to call. A kind of ToC of your project.

wsgi.py − If you need to deploy your project over WSGI.

**6.2.3 Setting Up Your Project**

Your project is set up in the subfolder myproject/settings.py. Following are some important options you might need to set −

DEBUG = True

This option lets you set if your project is in debug mode or not. Debug mode lets you get more information about your project's error. Never set it to ‘True’ for a live project. However, this has to be set to ‘True’ if you want the Django light server to serve static files. Do it only in the development mode.

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': 'database.sql',

'USER': '',

'PASSWORD': '',

'HOST': '',

'PORT': '',

}

}

Database is set in the ‘Database’ dictionary. The example above is for SQLite engine. As stated earlier, Django also supports −

MySQL (django.db.backends.mysql)

PostGreSQL (django.db.backends.postgresql\_psycopg2)

Oracle (django.db.backends.oracle) and NoSQL DB

MongoDB (django\_mongodb\_engine)

Before setting any new engine, make sure you have the correct db driver installed.

You can also set others options like: TIME\_ZONE, LANGUAGE\_CODE, TEMPLATE…

Now that your project is created and configured make sure it's working −

$ python manage.py runserver

You will get something like the following on running the above code −

Validating models...

0 errors found

September 03, 2015 - 11:41:50

Django version 1.6.11, using settings 'myproject.settings'

Starting development server at http://127.0.0.1:8000/

Quit the server with CONTROL-C.

A project is a sum of many applications. Every application has an objective and can be reused into another project, like the contact form on a website can be an application, and can be reused for others. See it as a module of your project.

**6.2.4 Create an Application**

We assume you are in your project folder. In our main “myproject” folder, the same folder then manage.py −

$ python manage.py startapp myapp

You just created myapp application and like project, Django create a “myapp” folder with the application structure −

myapp/

\_\_init\_\_.py

admin.py

models.py

tests.py

views.py

\_\_init\_\_.py − Just to make sure python handles this folder as a package.

admin.py − This file helps you make the app modifiable in the admin interface.

models.py − This is where all the application models are stored.

tests.py − This is where your unit tests are.

views.py − This is where your application views are.

**6.2.5 Get the Project to Know About Your Application**

At this stage we have our "myapp" application, now we need to register it with our Django project "myproject". To do so, update INSTALLED\_APPS tuple in the settings.py file of your project (add your app name) −

INSTALLED\_APPS = (

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'myapp',

)

Creating forms in Django, is really similar to creating a model. Here again, we just need to inherit from Django class and the class attributes will be the form fields. Let's add a forms.py file in myapp folder to contain our app forms. We will create a login form.

myapp/forms.py

#-\*- coding: utf-8 -\*-

from django import forms

class LoginForm(forms.Form):

user = forms.CharField(max\_length = 100)

password = forms.CharField(widget = forms.PasswordInput())

As seen above, the field type can take "widget" argument for html rendering; in our case, we want the password to be hidden, not displayed. Many others widget are present in Django: DateInput for dates, CheckboxInput for checkboxes, etc.

**6.2.6 Using Form in a View**

There are two kinds of HTTP requests, GET and POST. In Django, the request object passed as parameter to your view has an attribute called "method" where the type of the request is set, and all data passed via POST can be accessed via the request.POST dictionary.

Let's create a login view in our myapp/views.py −

#-\*- coding: utf-8 -\*-

from myapp.forms import LoginForm

def login(request):

username = "not logged in"

if request.method == "POST":

#Get the posted form

MyLoginForm = LoginForm(request.POST)

if MyLoginForm.is\_valid():

username = MyLoginForm.cleaned\_data['username']

else:

MyLoginForm = Loginform()

return render(request, 'loggedin.html', {"username" : username})

The view will display the result of the login form posted through the loggedin.html. To test it, we will first need the login form template. Let's call it login.html.

<html>

<body>

<form name = "form" action = "{% url "myapp.views.login" %}"

method = "POST" >{% csrf\_token %}

<div style = "max-width:470px;">

<center>

<input type = "text" style = "margin-left:20%;"

placeholder = "Identifiant" name = "username" />

</center>

</div>

<br>

<div style = "max-width:470px;">

<center>

<input type = "password" style = "margin-left:20%;"

placeholder = "password" name = "password" />

</center>

</div>

<br>

<div style = "max-width:470px;">

<center>

<button style = "border:0px; background-color:#4285F4; margin-top:8%;

height:35px; width:80%;margin-left:19%;" type = "submit" value = "Login" >

<strong>Login</strong>

</button>

</center>

</div>

</form>

</body>

</html>

The template will display a login form and post the result to our login view above. You have probably noticed the tag in the template, which is just to prevent Cross-site Request Forgery (CSRF) attack on your site.

{% csrf\_token %}

Once we have the login template, we need the loggedin.html template that will be rendered after form treatment.

<html>

<body>

You are : <strong>{{username}}</strong>

</body>

</html>

Now, we just need our pair of URLs to get started: myapp/urls.py

from django.conf.urls import patterns, url

from django.views.generic import TemplateView

urlpatterns = patterns('myapp.views',

url(r'^connection/',TemplateView.as\_view(template\_name = 'login.html')),

url(r'^login/', 'login', name = 'login'))

When accessing "/myapp/connection", we will get the following login.html template rendered −

**6.2.7 Setting Up Sessions**

In Django, enabling session is done in your project settings.py, by adding some lines to the MIDDLEWARE\_CLASSES and the INSTALLED\_APPS options. This should be done while creating the project, but it's always good to know, so MIDDLEWARE\_CLASSES should have −

'django.contrib.sessions.middleware.SessionMiddleware'

And INSTALLED\_APPS should have −

'django.contrib.sessions'

By default, Django saves session information in database (django\_session table or collection), but you can configure the engine to store information using other ways like: in file or in cache.

When session is enabled, every request (first argument of any view in Django) has a session (dict) attribute.

Let's create a simple sample to see how to create and save sessions. We have built a simple login system before (see Django form processing chapter and Django Cookies Handling chapter). Let us save the username in a cookie so, if not signed out, when accessing our login page you won’t see the login form. Basically, let's make our login system we used in Django Cookies handling more secure, by saving cookies server side.

For this, first lets change our login view to save our username cookie server side −

def login(request):

username = 'not logged in'

if request.method == 'POST':

MyLoginForm = LoginForm(request.POST)

if MyLoginForm.is\_valid():

username = MyLoginForm.cleaned\_data['username']

request.session['username'] = username

else:

MyLoginForm = LoginForm()

return render(request, 'loggedin.html', {"username" : username}

Then let us create formView view for the login form, where we won’t display the form if cookie is set −

def formView(request):

if request.session.has\_key('username'):

username = request.session['username']

return render(request, 'loggedin.html', {"username" : username})

else:

return render(request, 'login.html', {})

Now let us change the url.py file to change the url so it pairs with our new view −

from django.conf.urls import patterns, url

from django.views.generic import TemplateView

urlpatterns = patterns('myapp.views',

url(r'^connection/','formView', name = 'loginform'),

url(r'^login/', 'login', name = 'login'))

When accessing /myapp/connection, you will get to see the following page