Логистическая регрессия

Линейная регрессионная модель не всегда способна качественно предсказывать значения зависимой переменной. Выбирая для построения модели линейное уравнение, мы естественным образом не накладываем никаких ограничений на значения зависимой переменной. А такие ограничения могут быть существенными.

Например, при проектировании оптимальной длины шахты лифта в новом здании необходимо учесть, что эта длина не может превышать высоту здания вообще.

Линейная регрессионная модель может дать результаты, несовместимые с реальностью. С целью решения данных проблем полезно изменить вид уравнения регрессии и подстроить его для решения конкретной задачи.

Вообще, логит регрессионная модель предназначена для решения задач предсказания значения непрерывной зависимой переменной, при условии, что эта зависимая переменная может принимать значения на интервале от 0 до 1.

В силу такой специфики, ее часто используют для предсказания вероятности наступления некоторого события в зависимости от значений некоторого числа предикторов.

Можно использовать логит регрессию и для решения задач с бинарным откликом. Такие задачи появляются, когда зависимая переменная может принимать только два значения.

Приведем конкретный пример. Пусть требуется предсказать эффективность операции по пересадке сердца. Такие операции очень сложны и результата от их проведения может быть только два- пациент жив или умер (точнее, пережил ли он месяц после трансплантации - этот срок является определяющим).

В качестве предикторов используются данные предоперационного обследования и клинические параметры, например, возраст, уровень холестерина в крови, давление, группа крови и т.д. Задача свелась к классификации пациентов на две группы. Для первой группы прогноз положительный, для второй - отрицательный. Решение подобной задачи может повлиять на принятие решения о проведении операции - стоит ли вообще проводить пересадку, если вероятность пережить месяц после трансплантации для пациента невелика?

Математическая основа логистической регрессии

Итак, как уже было сказано, в логит регрессионной модели предсказанные значения зависимой переменной или переменной отклика не могут быть меньше (или равными) 0, или больше (или равными) 1, не зависимо от значений независимых переменных; поэтому, эта модель часто используется для анализа бинарных зависимых переменных или переменных отклика.

При этом используется следующее уравнение регрессии (термин логит был впервые использован Berkson, 1944):

$$y=\exp(b_0+b_1*x_1+...+b_n*x_n)/[1+\exp(b_0+b_1*x_1+...+b_n*x_n)]$$

Легко увидеть, что независимо от регрессионных коэффициентов или величин x, предсказанные значения (y) в этой модели всегда будут лежать в диапазоне от 0 до 1.

Термин логит произошел от того, что эту модель легко линеаризовать с помощью логит преобразования. Предположим, что бинарная зависимая переменная y является непрерывной вероятностью p, лежащей в диапазоне от 0 до 1. Тогда можно преобразовать эту вероятность p следующим образом:

$$p' = \log_e \{p/(1-p)\}$$

Это преобразование называется логит или логистическим преобразованием.

Заметим, что p' теоретически может принимать любые значения от минус до плюс бесконечности. Поскольку логит преобразование решает проблему 0/1 границ для исходной зависимой переменной (вероятности), то можно использовать эти (логит преобразованные) значения в обычном линейном уравнении регрессии.

Фактически, при проведении логит преобразования обеих частей логит регрессионного уравнения, приведенного выше, мы получим стандартную линейную модель множественной регрессии:

$$p' = b_0 + b_1 * x_1 + b_2 * x_2 + \dots + b_n * x_n$$

Подобное уравнение нам уже знакомо. Решив его, мы получим значения регрессионных коэффициентов, по которым затем можно восстановить вероятность p.

Особенности логит регрессии

Однако, применение логистического преобразования к уравнению логит регрессии порождает определенные проблемы.

При решении задачи линейной регрессии мы подгоняли к наблюдаемым значениям некоторую гиперповерхность - прямую в случае простой регрессии, плоскость - в случае двух независимых переменных. Также мы требуем нормальность и некоррелированность ошибок.

При переходе к уравнению логит регрессии подгоняемая поверхность уже не будет иметь такой простой вид. Также, нас не спасет уже и нормальность ошибок.

Все это делает невозможным использования методов оценивания, применяемых для линейных задач.

Например, в случае одной независимой переменной для простой регрессии применялся известный метод наименьших квадратов. В случае простой логит регрессии такой метод уже неприменим. Неприменимыми являются и подобные методы для решения задач с большим числом предикторов.

Поэтому для решения задач логит регрессии используется только **метод максимального правдоподобия.** Вкратце, процесс оценки регрессионных коэффициентов сводится к максимизации вероятности появления конкретной выборки (при заданных наблюдаемых значениях). Это приводит к часто невысокому проценту корректной классификации. Логит регрессия также слабо устойчива к излишней подгонке.

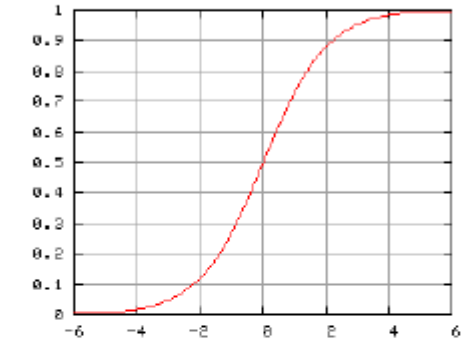
++

Цели:

Привести примеры использования логистической регрессии.

Логистическая регрессия или **логит регрессия** (англ. *logit model*) — это статистическая модель, используемая для предсказания вероятности возникновения некоторого события путём подгонки данных к логистической кривой.

Описание



Логистическая функция: $f(x) = \frac{1}{1 + e^{-x}}$.

Логистическая регрессия применяется для предсказания вероятности возникновения некоторого события по значениям множества признаков. Для этого вводится так называемая *зависимая переменная y*, принимающая лишь одно из двух значений — как правило, это числа 0 (событие не произошло) и 1 (событие произошло), и множество *независимых переменных* (также называемых признаками, предикторами или регрессорами) — вещественных $x_1,x_2,...,x_n$, на основе значений которых требуется вычислить вероятность принятия того или иного значения зависимой переменной.

Делается предположение о том, что вероятность наступления события $y = 1$ равна:

$$\Pr\{y = 1|x\} = f(z),$$

где $z = \theta^T x = \theta_1 x_1 + \ldots + \theta_n x_n$, x и θ — вектора-столбцы значений независимых переменных $x_1,...,x_n$ и параметров (коэффициентов регрессии) — вещественных чисел $\theta_1,...,\theta_n$, соответственно, а $f(z)$ — так называемая *логистическая функция* (иногда также называемая сигмоидом или логит-функцией):

$$f(z) = \frac{1}{1+e^{-z}}.$$

Так как y принимает лишь значения 0 и 1, то вероятность второго возможного значения равна:

$$\Pr\{y=0|x\} = 1 - f(z) = 1 - f(\theta^T x).$$

Для краткости, функцию распределения y при заданном x можно записать в таком виде:

$$\Pr\{y|x\} = f(\theta^T x)^y (1 - f(\theta^T x))^{1-y}, \quad y \in \{0,1\}.$$

Фактически, это есть распределение Бернулли с параметром, равным $f(\theta^T x)$.

Подбор параметров

Для подбора параметров $\theta_1,...,\theta_n$ необходимо составить обучающую выборку, состоящую из наборов значений независимых переменных и соответствующих им значений зависимой переменной y. Формально, это множество пар $(x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)})$, где $x^{(i)} \in \mathbb{R}^n$ — вектор значений независимых переменных, а $y^{(i)} \in \{0,1\}$ — соответствующее им значение y. Каждая такая пара называется обучающим примером.

Обычно используется метод максимального правдоподобия, согласно которому выбираются параметры θ, максимизирующие значение функции правдоподобия на обучающей выборке:

$$\hat{\theta} = \operatorname{argmax}_\theta L(\theta) = \operatorname{argmax}_\theta \prod_{i=1}^{m} \Pr\{y = y^{(i)}|x = x^{(i)}\}.$$

Максимизация функции правдоподобия эквивалентна максимизации её логарифма:

$$\log L(\theta) = \sum_{i=1}^{m} \log \Pr\{y = y^{(i)}|x = x^{(i)}\} = \sum_{i=1}^{m} y^{(i)} \log f(\theta^T x^{(i)}) + (1-y^{(i)}) \log($$

Для максимизации этой функции может быть применён, например, метод градиентного спуска. Он заключается в выполнении следующих итераций, начиная с некоторого начального значения параметров θ:

$$\theta := \theta + \alpha \nabla \log L(\theta) = \theta + \alpha \sum_{i=1}^{m} (y^{(i)} - f(\theta^T x^{(i)})) x^{(i)}, \quad \alpha > 0.$$

На практике также применяют метод Ньютона и стохастический градиентный спуск.

++
+

Reading and Writing CSV Files in Python

by Jon Fincher Jul 16, 2018 26 Comments **data-science** **intermediate** **python**

Table of Contents

Let's face it: you need to get information into and out of your programs through more than just the keyboard and console. Exchanging information through text files is a common way to share info between programs. One of the most popular formats for exchanging data is the CSV format. But how do you use it?

Let's get one thing clear: you don't have to (and you won't) build your own CSV parser from scratch. There are several perfectly acceptable libraries you can use. The Python `csv` library will work for most cases. If your work requires lots of data or numerical analysis, the `pandas` library has CSV parsing capabilities as well, which should handle the rest.

In this article, you'll learn how to read, process, and parse CSV from text files using Python. You'll see how CSV files work, learn the all-important `csv` library built into Python, and see how CSV parsing works using the `pandas` library.

So let's get started!

What Is a CSV File?

A CSV file (Comma Separated Values file) is a type of plain text file that uses specific structuring to arrange tabular data. Because it's a plain text file, it can contain only actual text data—in other words, printable ASCII or Unicode characters.

The structure of a CSV file is given away by its name. Normally, CSV files use a comma to separate each specific data value. Here's what that structure looks like:

```
column 1 name,column 2 name, column 3 name
first row data 1,first row data 2,first row data 3
second row data 1,second row data 2,second row data 3
...
```

Notice how each piece of data is separated by a comma. Normally, the first line identifies each piece of data—in other words, the name of a data column. Every subsequent line after that is actual data and is limited only by file size constraints.

In general, the separator character is called a delimiter, and the comma is not the only one used. Other popular delimiters include the tab (`\t`), colon (`:`) and semi-colon (`;`) characters. Properly parsing a CSV file requires us to know which delimiter is being used.

Where Do CSV Files Come From?

CSV files are normally created by programs that handle large amounts of data. They are a convenient way to export data from spreadsheets and databases as well as import or use it in other programs. For example, you might export the results of a data mining program to a CSV file and then import that into a spreadsheet to analyze the data, generate graphs for a presentation, or prepare a report for publication.

CSV files are very easy to work with programmatically. Any language that supports text file input and string manipulation (like Python) can work with CSV files directly.

Parsing CSV Files With Python's Built-in CSV Library

The `csv` library provides functionality to both read from and write to CSV files. Designed to work out of the box with Excel-generated CSV files, it is easily adapted to work with a variety of CSV formats. The `csv` library contains objects and other code to read, write, and process data from and to CSV files.

Reading CSV Files With `csv`

Reading from a CSV file is done using the `reader` object. The CSV file is opened as a text file with Python's built-in `open()` function, which returns a file object. This is then passed to the`reader`, which does the heavy lifting.

Here's the `employee_birthday.txt` file:

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

Here's code to read it:

```
import csv

with open('employee_birthday.txt') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            print(f'\t{row[0]} works in the {row[1]} department, and
was born in {row[2]}.')
            line_count += 1
    print(f'Processed {line_count} lines.')
```

This results in the following output:

```
Column names are name, department, birthday month
    John Smith works in the Accounting department, and was born in
November.
    Erica Meyers works in the IT department, and was born in March.
Processed 3 lines.
```

Each row returned by the `reader` is a list of `String` elements containing the data found by removing the delimiters. The first row returned contains the column names, which is handled in a special way.

Reading CSV Files Into a Dictionary With `csv`

Rather than deal with a list of individual `String` elements, you can read CSV data directly into a dictionary (technically, an Ordered Dictionary) as well.

Again, our input file, `employee_birthday.txt` is as follows:

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

Here's the code to read it in as a dictionary this time:

```
import csv

with open('employee_birthday.txt', mode='r') as csv_file:
    csv_reader = csv.DictReader(csv_file)
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        print(f'\t{row["name"]} works in the {row["department"]}
department, and was born in {row["birthday month"]}.')
        line_count += 1
    print(f'Processed {line_count} lines.')
```

This results in the same output as before:

```
Column names are name, department, birthday month
    John Smith works in the Accounting department, and was born in
November.
    Erica Meyers works in the IT department, and was born in March.
Processed 3 lines.
```

Where did the dictionary keys come from? The first line of the CSV file is assumed to contain the keys to use to build the dictionary. If you don't have these in your CSV file, you should specify your own keys by setting the `fieldnames` optional parameter to a list containing them.

Optional Python CSV `reader` Parameters

The `reader` object can handle different styles of CSV files by specifying additional parameters, some of which are shown below:

- `delimiter` specifies the character used to separate each field. The default is the comma (`','`).

- `quotechar` specifies the character used to surround fields that contain the delimiter character. The default is a double quote (`' " '`).
- `escapechar` specifies the character used to escape the delimiter character, in case quotes aren't used. The default is no escape character.

These parameters deserve some more explanation. Suppose you're working with the following `employee_addresses.txt` file:

```
name,address,date joined
john smith,1132 Anywhere Lane Hoboken NJ, 07030,Jan 4
erica meyers,1234 Smith Lane Hoboken NJ, 07030,March 2
```

This CSV file contains three fields: `name`, `address`, and `date joined`, which are delimited by commas. The problem is that the data for the `address` field also contains a comma to signify the zip code.

There are three different ways to handle this situation:

- **Use a different delimiter**
  That way, the comma can safely be used in the data itself. You use the `delimiter`optional parameter to specify the new delimiter.
- **Wrap the data in quotes**
  The special nature of your chosen delimiter is ignored in quoted strings. Therefore, you can specify the character used for quoting with the `quotechar` optional parameter. As long as that character also doesn't appear in the data, you're fine.
- **Escape the delimiter characters in the data**
  Escape characters work just as they do in format strings, nullifying the interpretation of the character being escaped (in this case, the delimiter). If an escape character is used, it must be specified using the `escapechar` optional parameter.

Writing CSV Files With `csv`

You can also write to a CSV file using a `writer` object and the `.write_row()` method:

```
import csv
```

```
with open('employee_file.csv', mode='w') as employee_file:
    employee_writer = csv.writer(employee_file, delimiter=',',
quotechar='"', quoting=csv.QUOTE_MINIMAL)

    employee_writer.writerow(['John Smith', 'Accounting', 'November'])
    employee_writer.writerow(['Erica Meyers', 'IT', 'March'])
```

The `quotechar` optional parameter tells the `writer` which character to use to quote fields when writing. Whether quoting is used or not, however, is determined by the `quoting`optional parameter:

- If `quoting` is set to `csv.QUOTE_MINIMAL`, then `.writerow()` will quote fields only if they contain the `delimiter` or the `quotechar`. This is the default case.
- If `quoting` is set to `csv.QUOTE_ALL`, then `.writerow()` will quote all fields.
- If `quoting` is set to `csv.QUOTE_NONNUMERIC`, then `.writerow()` will quote all fields containing text data and convert all numeric fields to the `float` data type.
- If `quoting` is set to `csv.QUOTE_NONE`, then `.writerow()` will escape delimiters instead of quoting them. In this case, you also must provide a value for the `escapechar` optional parameter.

Reading the file back in plain text shows that the file is created as follows:

```
John Smith,Accounting,November
Erica Meyers,IT,March
```

Writing CSV File From a Dictionary With `csv`

Since you can read our data into a dictionary, it's only fair that you should be able to write it out from a dictionary as well:

```
import csv

with open('employee_file2.csv', mode='w') as csv_file:
    fieldnames = ['emp_name', 'dept', 'birth_month']
    writer = csv.DictWriter(csv_file, fieldnames=fieldnames)

    writer.writeheader()
```

```
    writer.writerow({'emp_name': 'John Smith', 'dept': 'Accounting',
'birth_month': 'November'})
    writer.writerow({'emp_name': 'Erica Meyers', 'dept': 'IT',
'birth_month': 'March'})
```

Unlike `DictReader`, the `fieldnames` parameter is required when writing a dictionary. This makes sense, when you think about it: without a list of `fieldnames`, the `DictWriter` can't know which keys to use to retrieve values from your dictionaries. It also uses the keys in`fieldnames` to write out the first row as column names.

The code above generates the following output file:

```
emp_name,dept,birth_month
John Smith,Accounting,November
Erica Meyers,IT,March
```

Parsing CSV Files With the `pandas` Library

Of course, the Python CSV library isn't the only game in town. Reading CSV files is possible in `pandas` as well. It is highly recommended if you have a lot of data to analyze.

`pandas` is an open-source Python library that provides high performance data analysis tools and easy to use data structures. `pandas` is available for all Python installations, but it is a key part of the Anaconda distribution and works extremely well in Jupyter notebooks to share data, code, analysis results, visualizations, and narrative text.

Installing `pandas` and its dependencies in `Anaconda` is easily done:

```
$ conda install pandas
```

As is using `pip`/`pipenv` for other Python installations:

```
$ pip install pandas
```

We won't delve into the specifics of how `pandas` works or how to use it. For an in-depth treatment on using `pandas` to read and analyze large data sets, check out Shantnu Tiwari'ssuperb article on working with large Excel files in pandas.

Reading CSV Files With `pandas`

To show some of the power of `pandas` CSV capabilities, I've created a slightly more complicated file to read, called `hrdata.csv`. It contains data on company employees:

```
Name,Hire Date,Salary,Sick Days remaining
Graham Chapman,03/15/14,50000.00,10
John Cleese,06/01/15,65000.00,8
Eric Idle,05/12/14,45000.00,10
Terry Jones,11/01/13,70000.00,3
Terry Gilliam,08/12/14,48000.00,7
Michael Palin,05/23/13,66000.00,8
```

Reading the CSV into a `pandas DataFrame` is quick and straightforward:

```
import pandas
df = pandas.read_csv('hrdata.csv')
print(df)
```

That's it: three lines of code, and only one of them is doing the actual work.`pandas.read_csv()` opens, analyzes, and reads the CSV file provided, and stores the data in aDataFrame. Printing the `DataFrame` results in the following output:

```
             Name Hire Date   Salary  Sick Days remaining
0  Graham Chapman  03/15/14  50000.0                   10
1     John Cleese  06/01/15  65000.0                    8
2       Eric Idle  05/12/14  45000.0                   10
3     Terry Jones  11/01/13  70000.0                    3
4   Terry Gilliam  08/12/14  48000.0                    7
5   Michael Palin  05/23/13  66000.0                    8
```

Here are a few points worth noting:

- First, `pandas` recognized that the first line of the CSV contained column names, and used them automatically. I call this Goodness.
- However, `pandas` is also using zero-based integer indices in the `DataFrame`. That's because we didn't tell it what our index should be.
- Further, if you look at the data types of our columns , you'll see `pandas` has properly converted the `Salary` and `Sick Days remaining` columns to numbers, but the `Hire Date`column is still a `String`. This is easily confirmed in interactive mode:

```
>>>
```

```
>>> print(type(df['Hire Date'][0]))
<class 'str'>
```

Let's tackle these issues one at a time. To use a different column as the `DataFrame` index, add the `index_col` optional parameter:

```
import pandas
df = pandas.read_csv('hrdata.csv', index_col='Name')
print(df)
```

Now the `Name` field is our `DataFrame` index:

```
                Hire Date   Salary  Sick Days remaining
Name
Graham Chapman  03/15/14  50000.0                   10
John Cleese     06/01/15  65000.0                    8
Eric Idle       05/12/14  45000.0                   10
Terry Jones     11/01/13  70000.0                    3
Terry Gilliam   08/12/14  48000.0                    7
Michael Palin   05/23/13  66000.0                    8
```

Next, let's fix the data type of the `Hire Date` field. You can force `pandas` to read data as a date with the `parse_dates` optional parameter, which is defined as a list of column names to treat as dates:

```
import pandas
df = pandas.read_csv('hrdata.csv', index_col='Name',
parse_dates=['Hire Date'])
print(df)
```

Notice the difference in the output:

```
                Hire Date   Salary  Sick Days remaining
Name
Graham Chapman 2014-03-15  50000.0                   10
John Cleese    2015-06-01  65000.0                    8
Eric Idle      2014-05-12  45000.0                   10
Terry Jones    2013-11-01  70000.0                    3
Terry Gilliam  2014-08-12  48000.0                    7
Michael Palin  2013-05-23  66000.0                    8
```

The date is now formatted properly, which is easily confirmed in interactive mode:

```
>>>
```

```
>>> print(type(df['Hire Date'][0]))
<class 'pandas._libs.tslibs.timestamps.Timestamp'>
```

If your CSV files doesn't have column names in the first line, you can use the `names` optional parameter to provide a list of column names. You can also use this if you want to override the column names provided in the first line. In this case, you must also tell`pandas.read_csv()` to ignore existing column names using the `header=0` optional parameter:

```
import pandas
df = pandas.read_csv('hrdata.csv',
            index_col='Employee',
            parse_dates=['Hired'],
            header=0,
            names=['Employee', 'Hired','Salary', 'Sick Days'])
print(df)
```

Notice that, since the column names changed, the columns specified in the `index_col` and`parse_dates` optional parameters must also be changed. This now results in the following output:

```
                    Hired   Salary  Sick Days
Employee
Graham Chapman 2014-03-15  50000.0         10
John Cleese    2015-06-01  65000.0          8
Eric Idle      2014-05-12  45000.0         10
Terry Jones    2013-11-01  70000.0          3
Terry Gilliam  2014-08-12  48000.0          7
Michael Palin  2013-05-23  66000.0          8
```

Writing CSV Files With `pandas`

Of course, if you can't get your data out of `pandas` again, it doesn't do you much good. Writing a `DataFrame` to a CSV file is just as easy as reading one in. Let's write the data with the new column names to a new CSV file:

```
import pandas
df = pandas.read_csv('hrdata.csv',
            index_col='Employee',
            parse_dates=['Hired'],
            header=0,
            names=['Employee', 'Hired', 'Salary', 'Sick Days'])
```

```
df.to_csv('hrdata_modified.csv')
```

The only difference between this code and the reading code above is that the `print(df)` call was replaced with `df.to_csv()`, providing the file name. The new CSV file looks like this:

```
Employee,Hired,Salary,Sick Days
Graham Chapman,2014-03-15,50000.0,10
John Cleese,2015-06-01,65000.0,8
Eric Idle,2014-05-12,45000.0,10
Terry Jones,2013-11-01,70000.0,3
Terry Gilliam,2014-08-12,48000.0,7
Michael Palin,2013-05-23,66000.0,8
```

Conclusion

If you understand the basics of reading CSV files, then you won't ever be caught flat footed when you need to deal with importing data. Most CSV reading, processing, and writing tasks can be easily handled by the basic `csv` Python library. If you have a lot of data to read and process, the `pandas` library provides quick and easy CSV handling capabilities as well.

Are there other ways to parse text files? Of course! Libraries like ANTLR, PLY, and PlyPluscan all handle heavy-duty parsing, and if simple `String` manipulation won't work, there are always regular expressions.

But those are topics for other articles…

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++

# This Python 3 environment comes with many helpful analytics libraries
installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/
docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will
list the files in the input directory
```

```
#import os
#print(os.listdir("../input"))
dataset = pd.read_csv('../input/Social_Network_Ads.csv')

# Any results you write to the current directory are saved as output.
```

In [2]:

```
dataset
```

Out[2]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

In [3]:

```
dataset.shape
```

Out[3]:

```
(400, 5)
```

In [4]:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
User ID            400 non-null int64
Gender             400 non-null object
Age                400 non-null int64
EstimatedSalary    400 non-null int64
Purchased          400 non-null int64
dtypes: int64(4), object(1)
memory usage: 15.7+ KB
```

In [5]:

```
dataset.columns
```

Out[5]:

```
Index(['User ID', 'Gender', 'Age', 'EstimatedSalary', 'Purchased'],
dtype='object')
```

In [6]:

```
x = dataset.iloc[:,2:4].values
y= dataset.iloc[:,4].values
```

In [7]:

```
# Split the data into Training and Testing set
from sklearn.cross_validation import train_test_split
x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.25,random_state=0)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/cross_validation.py:41:
DeprecationWarning: This module was deprecated in version 0.18 in favor of
the model_selection module into which all the refactored classes and
functions are moved. Also note that the interface of the new CV iterators
are different from that of this module. This module will be removed in
0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [8]:

```
# Feature scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [9]:

```
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

In [10]:

```
x_train
```

Out[10]:

```
array([[ 0.58164944, -0.88670699],
       [-0.60673761,  1.46173768],
       [-0.01254409, -0.5677824 ],
       [-0.60673761,  1.89663484],
       [ 1.37390747, -1.40858358],
       [ 1.47293972,  0.99784738],
       [ 0.08648817, -0.79972756],
       [-0.01254409, -0.24885782],
```

```
       [-0.21060859, -0.5677824 ],
       [-0.21060859, -0.19087153],
       [-0.30964085, -1.29261101],
       [-0.30964085, -0.5677824 ],
       [ 0.38358493,  0.09905991],
       [ 0.8787462 , -0.59677555],
       [ 2.06713324, -1.17663843],
       [ 1.07681071, -0.13288524],
       [ 0.68068169,  1.78066227],
       [-0.70576986,  0.56295021],
       [ 0.77971394,  0.35999821],
       [ 0.8787462 , -0.53878926],
       [-1.20093113, -1.58254245],
       [ 2.1661655 ,  0.93986109],
       [-0.01254409,  1.22979253],
       [ 0.18552042,  1.08482681],
       [ 0.38358493, -0.48080297],
       [-0.30964085, -0.30684411],
       [ 0.97777845, -0.8287207 ],
       [ 0.97777845,  1.8676417 ],
       [-0.01254409,  1.25878567],
       [-0.90383437,  2.27354572],
       [-1.20093113, -1.58254245],
       [ 2.1661655 , -0.79972756],
       [-1.39899564, -1.46656987],
       [ 0.38358493,  2.30253886],
       [ 0.77971394,  0.76590222],
       [-1.00286662, -0.30684411],
       [ 0.08648817,  0.76590222],
       [-1.00286662,  0.56295021],
       [ 0.28455268,  0.07006676],
       [ 0.68068169, -1.26361786],
       [-0.50770535, -0.01691267],
       [-1.79512465,  0.35999821],
       [-0.70576986,  0.12805305],
       [ 0.38358493,  0.30201192],
       [-0.30964085,  0.07006676],
       [-0.50770535,  2.30253886],
       [ 0.18552042,  0.04107362],
       [ 1.27487521,  2.21555943],
       [ 0.77971394,  0.27301877],
       [-0.30964085,  0.1570462 ],
       [-0.01254409, -0.53878926],
       [-0.21060859,  0.1570462 ],
       [-0.11157634,  0.24402563],
       [-0.01254409, -0.24885782],
       [ 2.1661655 ,  1.11381995],
       [-1.79512465,  0.35999821],
       [ 1.86906873,  0.12805305],
       [ 0.38358493, -0.13288524],
       [-1.20093113,  0.30201192],
       [ 0.77971394,  1.37475825],
       [-0.30964085, -0.24885782],
       [-1.6960924 , -0.04590581],
       [-1.00286662, -0.74174127],
       [ 0.28455268,  0.50496393],
```

```
       [-0.11157634, -1.06066585],
       [-1.10189888,  0.59194336],
       [ 0.08648817, -0.79972756],
       [-1.00286662,  1.54871711],
       [-0.70576986,  1.40375139],
       [-1.29996338,  0.50496393],
       [-0.30964085,  0.04107362],
       [-0.11157634,  0.01208048],
       [-0.30964085, -0.88670699],
       [ 0.8787462 , -1.3505973 ],
       [-0.30964085,  2.24455257],
       [ 0.97777845,  1.98361427],
       [-1.20093113,  0.47597078],
       [-1.29996338,  0.27301877],
       [ 1.37390747,  1.98361427],
       [ 1.27487521, -1.3505973 ],
       [-0.30964085, -0.27785096],
       [-0.50770535,  1.25878567],
       [-0.80480212,  1.08482681],
       [ 0.97777845, -1.06066585],
       [ 0.28455268,  0.30201192],
       [ 0.97777845,  0.76590222],
       [-0.70576986, -1.49556302],
       [-0.70576986,  0.04107362],
       [ 0.48261718,  1.72267598],
       [ 2.06713324,  0.18603934],
       [-1.99318916, -0.74174127],
       [-0.21060859,  1.40375139],
       [ 0.38358493,  0.59194336],
       [ 0.8787462 , -1.14764529],
       [-1.20093113, -0.77073441],
       [ 0.18552042,  0.24402563],
       [ 0.77971394, -0.30684411],
       [ 2.06713324, -0.79972756],
       [ 0.77971394,  0.12805305],
       [-0.30964085,  0.6209365 ],
       [-1.00286662, -0.30684411],
       [ 0.18552042, -0.3648304 ],
       [ 2.06713324,  2.12857999],
       [ 1.86906873, -1.26361786],
       [ 1.37390747, -0.91570013],
       [ 0.8787462 ,  1.25878567],
       [ 1.47293972,  2.12857999],
       [-0.30964085, -1.23462472],
       [ 1.96810099,  0.91086794],
       [ 0.68068169, -0.71274813],
       [-1.49802789,  0.35999821],
       [ 0.77971394, -1.3505973 ],
       [ 0.38358493, -0.13288524],
       [-1.00286662,  0.41798449],
       [-0.01254409, -0.30684411],
       [-1.20093113,  0.41798449],
       [-0.90383437, -1.20563157],
       [-0.11157634,  0.04107362],
       [-1.59706014, -0.42281668],
       [ 0.97777845, -1.00267957],
```

```
       [ 1.07681071, -1.20563157],
       [-0.01254409, -0.13288524],
       [-1.10189888, -1.52455616],
       [ 0.77971394, -1.20563157],
       [ 0.97777845,  2.07059371],
       [-1.20093113, -1.52455616],
       [-0.30964085,  0.79489537],
       [ 0.08648817, -0.30684411],
       [-1.39899564, -1.23462472],
       [-0.60673761, -1.49556302],
       [ 0.77971394,  0.53395707],
       [-0.30964085, -0.33583725],
       [ 1.77003648, -0.27785096],
       [ 0.8787462 , -1.03167271],
       [ 0.18552042,  0.07006676],
       [-0.60673761,  0.8818748 ],
       [-1.89415691, -1.40858358],
       [-1.29996338,  0.59194336],
       [-0.30964085,  0.53395707],
       [-1.00286662, -1.089659  ],
       [ 1.17584296, -1.43757673],
       [ 0.18552042, -0.30684411],
       [ 1.17584296, -0.74174127],
       [-0.30964085,  0.07006676],
       [ 0.18552042,  2.09958685],
       [ 0.77971394, -1.089659  ],
       [ 0.08648817,  0.04107362],
       [-1.79512465,  0.12805305],
       [-0.90383437,  0.1570462 ],
       [-0.70576986,  0.18603934],
       [ 0.8787462 , -1.29261101],
       [ 0.18552042, -0.24885782],
       [-0.4086731 ,  1.22979253],
       [-0.01254409,  0.30201192],
       [ 0.38358493,  0.1570462 ],
       [ 0.8787462 , -0.65476184],
       [ 0.08648817,  0.1570462 ],
       [-1.89415691, -1.29261101],
       [-0.11157634,  0.30201192],
       [-0.21060859, -0.27785096],
       [ 0.28455268, -0.50979612],
       [-0.21060859,  1.6067034 ],
       [ 0.97777845, -1.17663843],
       [-0.21060859,  1.63569655],
       [ 1.27487521,  1.8676417 ],
       [-1.10189888, -0.3648304 ],
       [-0.01254409,  0.04107362],
       [ 0.08648817, -0.24885782],
       [-1.59706014, -1.23462472],
       [-0.50770535, -0.27785096],
       [ 0.97777845,  0.12805305],
       [ 1.96810099, -1.3505973 ],
       [ 1.47293972,  0.07006676],
       [-0.60673761,  1.37475825],
       [ 1.57197197,  0.01208048],
       [-0.80480212,  0.30201192],
```

```
       [ 1.96810099,  0.73690908],
       [-1.20093113, -0.50979612],
       [ 0.68068169,  0.27301877],
       [-1.39899564, -0.42281668],
       [ 0.18552042,  0.1570462 ],
       [-0.50770535, -1.20563157],
       [ 0.58164944,  2.01260742],
       [-1.59706014, -1.49556302],
       [-0.50770535, -0.53878926],
       [ 0.48261718,  1.83864855],
       [-1.39899564, -1.089659  ],
       [ 0.77971394, -1.37959044],
       [-0.30964085, -0.42281668],
       [ 1.57197197,  0.99784738],
       [ 0.97777845,  1.43274454],
       [-0.30964085, -0.48080297],
       [-0.11157634,  2.15757314],
       [-1.49802789, -0.1038921 ],
       [-0.11157634,  1.95462113],
       [-0.70576986, -0.33583725],
       [-0.50770535, -0.8287207 ],
       [ 0.68068169, -1.37959044],
       [-0.80480212, -1.58254245],
       [-1.89415691, -1.46656987],
       [ 1.07681071,  0.12805305],
       [ 0.08648817,  1.51972397],
       [-0.30964085,  0.09905991],
       [ 0.08648817,  0.04107362],
       [-1.39899564, -1.3505973 ],
       [ 0.28455268,  0.07006676],
       [-0.90383437,  0.38899135],
       [ 1.57197197, -1.26361786],
       [-0.30964085, -0.74174127],
       [-0.11157634,  0.1570462 ],
       [-0.90383437, -0.65476184],
       [-0.70576986, -0.04590581],
       [ 0.38358493, -0.45180983],
       [-0.80480212,  1.89663484],
       [ 1.37390747,  1.28777882],
       [ 1.17584296, -0.97368642],
       [ 1.77003648,  1.83864855],
       [-0.90383437, -0.24885782],
       [-0.80480212,  0.56295021],
       [-1.20093113, -1.5535493 ],
       [-0.50770535, -1.11865214],
       [ 0.28455268,  0.07006676],
       [-0.21060859, -1.06066585],
       [ 1.67100423,  1.6067034 ],
       [ 0.97777845,  1.78066227],
       [ 0.28455268,  0.04107362],
       [-0.80480212, -0.21986468],
       [-0.11157634,  0.07006676],
       [ 0.28455268, -0.19087153],
       [ 1.96810099, -0.65476184],
       [-0.80480212,  1.3457651 ],
       [-1.79512465, -0.59677555],
```

```
       [-0.11157634,  0.12805305],
       [ 0.28455268, -0.30684411],
       [ 1.07681071,  0.56295021],
       [-1.00286662,  0.27301877],
       [ 1.47293972,  0.35999821],
       [ 0.18552042, -0.3648304 ],
       [ 2.1661655 , -1.03167271],
       [-0.30964085,  1.11381995],
       [-1.6960924 ,  0.07006676],
       [-0.01254409,  0.04107362],
       [ 0.08648817,  1.05583366],
       [-0.11157634, -0.3648304 ],
       [-1.20093113,  0.07006676],
       [-0.30964085, -1.3505973 ],
       [ 1.57197197,  1.11381995],
       [-0.80480212, -1.52455616],
       [ 0.08648817,  1.8676417 ],
       [-0.90383437, -0.77073441],
       [-0.50770535, -0.77073441],
       [-0.30964085, -0.91570013],
       [ 0.28455268, -0.71274813],
       [ 0.28455268,  0.07006676],
       [ 0.08648817,  1.8676417 ],
       [-1.10189888,  1.95462113],
       [-1.6960924 , -1.5535493 ],
       [-1.20093113, -1.089659  ],
       [-0.70576986, -0.1038921 ],
       [ 0.08648817,  0.09905991],
       [ 0.28455268,  0.27301877],
       [ 0.8787462 , -0.5677824 ],
       [ 0.28455268, -1.14764529],
       [-0.11157634,  0.67892279],
       [ 2.1661655 , -0.68375498],
       [-1.29996338, -1.37959044],
       [-1.00286662, -0.94469328],
       [-0.01254409, -0.42281668],
       [-0.21060859, -0.45180983],
       [-1.79512465, -0.97368642],
       [ 1.77003648,  0.99784738],
       [ 0.18552042, -0.3648304 ],
       [ 0.38358493,  1.11381995],
       [-1.79512465, -1.3505973 ],
       [ 0.18552042, -0.13288524],
       [ 0.8787462 , -1.43757673],
       [-1.99318916,  0.47597078],
       [-0.30964085,  0.27301877],
       [ 1.86906873, -1.06066585],
       [-0.4086731 ,  0.07006676],
       [ 1.07681071, -0.88670699],
       [-1.10189888, -1.11865214],
       [-1.89415691,  0.01208048],
       [ 0.08648817,  0.27301877],
       [-1.20093113,  0.33100506],
       [-1.29996338,  0.30201192],
       [-1.00286662,  0.44697764],
       [ 1.67100423, -0.88670699],
```

```
       [ 1.17584296,  0.53395707],
       [ 1.07681071,  0.53395707],
       [ 1.37390747,  2.331532  ],
       [-0.30964085, -0.13288524],
       [ 0.38358493, -0.45180983],
       [-0.4086731 , -0.77073441],
       [-0.11157634, -0.50979612],
       [ 0.97777845, -1.14764529],
       [-0.90383437, -0.77073441],
       [-0.21060859, -0.50979612],
       [-1.10189888, -0.45180983],
       [-1.20093113,  1.40375139]])
```

In [11]:

```
x_test
```

Out[11]:

```
array([[-0.54748976,  0.5130727 ],
       [ 0.15442019, -0.61825566],
       [-0.10879604,  0.14615539],
       [-0.54748976,  0.26846116],
       [-0.10879604, -0.61825566],
       [-0.81070599, -1.53554892],
       [-0.45975102, -1.68843113],
       [-0.0210573 ,  2.25592989],
       [-1.60035469, -0.0678797 ],
       [ 0.94406888, -0.83229075],
       [-0.54748976, -0.6488321 ],
       [-0.72296725, -0.46537345],
       [ 0.06668145, -0.46537345],
       [ 0.24215893,  0.20730828],
       [-1.4248772 ,  0.48249625],
       [-0.37201227,  1.43036596],
       [ 0.06668145,  0.20730828],
       [-1.51261594,  0.45191981],
       [ 1.64597884,  1.8278597 ],
       [-0.10879604, -1.47439603],
       [-0.10879604, -0.70998498],
       [ 0.94406888,  2.25592989],
       [ 0.41763642, -0.58767922],
       [ 0.94406888,  1.06344865],
       [-1.16166097, -1.29093738],
       [ 1.11954637,  2.16420057],
       [-0.72296725,  0.5130727 ],
       [-0.63522851,  0.2990376 ],
       [ 0.06668145, -0.25133835],
       [-0.37201227,  0.48249625],
       [-1.33713846,  0.54364914],
       [ 0.06668145,  0.26846116],
       [ 1.82145632, -0.31249124],
       [ 0.06668145, -0.52652633],
       [-1.07392223, -0.37364412],
       [-1.60035469, -0.55710277],
       [-1.24939971,  0.32961404],
       [-0.19653479, -0.83229075],
       [-0.45975102, -1.10747873],
       [ 1.11954637, -1.04632585],
```

```
       [-0.81070599,  0.54364914],
       [ 0.41763642, -0.55710277],
       [-0.81070599,  0.42134337],
       [-0.10879604, -1.53554892],
       [ 0.59311391,  1.27748375],
       [-0.81070599, -0.37364412],
       [ 0.06668145,  0.2990376 ],
       [ 1.3827626 ,  0.60480202],
       [-0.89844474, -1.2297845 ],
       [ 1.11954637,  0.48249625],
       [ 1.82145632,  1.58324817],
       [-0.19653479, -1.38266671],
       [-0.10879604, -0.40422056],
       [-0.19653479,  1.36921307],
       [ 1.99693381,  0.54364914],
       [ 0.7685914 , -1.16863161],
       [-0.63522851,  0.39076693],
       [-0.89844474,  0.2990376 ],
       [ 1.11954637, -1.29093738],
       [-1.16166097, -1.53554892],
       [-0.37201227, -1.5967018 ],
       [ 2.08467255, -0.86286719],
       [-1.51261594,  0.17673183],
       [-0.0210573 ,  0.87999   ],
       [-1.51261594, -1.35209027],
       [ 2.08467255,  0.39076693],
       [-1.07392223,  0.57422558],
       [-0.81070599, -0.37364412],
       [ 0.32989768, -0.70998498],
       [ 0.50537516, -0.00672682],
       [-0.37201227,  2.43938854],
       [-0.10879604,  0.20730828],
       [-1.24939971, -0.22076191],
       [ 0.7685914 , -1.47439603],
       [-0.81070599,  0.57422558],
       [-1.60035469,  0.36019049],
       [ 0.50537516,  0.26846116],
       [ 0.32989768, -0.31249124],
       [ 1.47050135, -1.10747873],
       [ 0.94406888,  1.12460154],
       [ 1.90919507,  2.25592989],
       [ 1.99693381,  0.39076693],
       [-1.07392223, -0.46537345],
       [-0.89844474, -1.07690229],
       [ 1.90919507, -0.98517296],
       [ 0.50537516,  0.2990376 ],
       [ 0.32989768,  0.14615539],
       [ 1.99693381,  1.8278597 ],
       [ 0.85633014, -0.89344364],
       [ 0.41763642, -0.31249124],
       [ 0.50537516, -0.19018547],
       [ 0.06668145,  2.31708278],
       [-1.16166097, -0.67940854],
       [-0.98618348, -1.13805517],
       [-1.07392223,  0.42134337],
       [-0.81070599,  0.78826068],
```

```
       [-1.16166097, -0.22076191],
       [ 1.03180763, -1.13805517],
       [ 1.03180763,  0.60480202],
       [ 0.50537516,  1.03287221]])
```

Feature scalling has done

- Next do the prediction

In [12]:

```
#Fitting logistic regression to the training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(x_train,y_train)
```

Out[12]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=0, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [13]:

```
# Predicting the Test set results
y_pred = classifier.predict(x_test)
```

In [14]:

```
y_pred
```

Out[14]:

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1])
```

Confusion matrix

- **Track prediction accuracy results of True positive vs True Negative (Sensitivity vs Specificity)**

In [15]:

```
# Making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
```

In [16]:

```
cm
```

Out[16]:

```
array([[63,  5],
       [ 7, 25]])
```

In [17]:

```
(63+25)/(63+7+25+5)
```

Out[17]:

```
0.88
```

To Get Automatic Confusion matrix accuracy result from below technique

In [18]:

```
from sklearn import metrics
```

```
print (metrics.accuracy_score(y_test,classifier.predict(x_test)))
```

```
0.88
```

Improvement Technique for Logistic Regression Use Case

- ROC Curve analysis to find an optimal cut_off point. We will try to Max(True Positive Rate, (1- False Positive Rate))
- Confusion Matrix building and Accuracy score to check the model performance

In [19]:

```
predict_prob_df=pd.DataFrame(classifier.predict_proba(x_test))
predict_prob_df.head(5)
fpr, tpr, thresholds = metrics.roc_curve( y_test,
                                          predict_prob_df[1],
                                          drop_intermediate = False )
thres=0
cut_off=0
for i in range(len(fpr)):
    if thres< ((1-fpr[i])+tpr[i]):
        thres=((1-fpr[i])+tpr[i])
        cut_off=thresholds[i]


# i am doing ROC curve analysis to find the best cut_off probablity for the
categorization.
#cut_off would contain the best cut_off point
#logic for cut_off selection Max(cut_off, (tpr+(1-fpr)))
#maximum sensitivity +specificity

predict_prob_df['final_label']=0
for k in range(len(predict_prob_df[1])):
    if predict_prob_df[1][k]>cut_off:
        predict_prob_df['final_label'][k]=1
    else:
        predict_prob_df['final_label'][k]=0


cm_1=confusion_matrix(y_test,predict_prob_df['final_label'],[1,0])
cm_1
```

Out[19]:

```
array([[26,  6],
       [ 5, 63]])
```

In [20]:

```
metrics.accuracy_score(y_test,predict_prob_df['final_label'])
# we have acheived the 89% accuracy, which is slight improvement of 1%.
```

Out[20]:

```
0.89
```

We have achived around 89% of accuracy from the previous prediction, looks like we have slightly improved by 1% fromt he previous prediction. Awesome one !!!!

Decision Tree Classifier

It is go to model/classification method for most of the dataset. The splitting criterion can be explicitely set Gini or entropy- information gain.

In [21]:

```
from sklearn.tree import DecisionTreeClassifier
clf_tree=DecisionTreeClassifier()
```

```
clf_tree.fit(x_train,y_train)
cm_2=confusion_matrix(y_test,clf_tree.predict(x_test),[1,0])
cm_2
```

Out[21]:

```
array([[28,  4],
       [ 7, 61]])
```

In [22]:

```
metrics.accuracy_score(y_test,clf_tree.predict(x_test))
#model accuracy is 89% same accuracy like previous prediction
```

Out[22]:

```
0.89
```

In [23]:

```
(28+61)/(28+7+61+4)
```

Out[23]:

```
0.89
```

SVM- Support Vector Machine for Classification

to identify the optimal parameters specification for the model for a given algorithm.

In [24]:

```
# Support Vector Machines
from sklearn.svm import SVC
svc = SVC()
svc.fit(x_train, y_train)
y_pred = svc.predict(x_test)
acc_svc = round(svc.score(x_train, y_train) * 100, 2)
acc_svc
```

Out[24]:

```
91.33
```

In [25]:

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(x_train, y_train)
y_pred = logreg.predict(x_test)
acc_log = round(logreg.score(x_train, y_train) * 100, 2)
acc_log
```

Out[25]:

```
82.33
```

In [26]:

```
# Linear SVC
from sklearn.svm import LinearSVC
linear_svc = LinearSVC()
linear_svc.fit(x_train, y_train)
y_pred = linear_svc.predict(x_test)
acc_linear_svc = round(linear_svc.score(x_train, y_train) * 100, 2)
acc_linear_svc
```

Out[26]:

```
82.67
```

- **Support vector machine has provided awesome prediction. SVM model has improved around 2% accuracy from the previos prediction**

KNN - (K-Nearest Neighbour Prediction)

In [27]:

```
#KNN - K-Nearest Neighbour
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(x_train, y_train)
y_pred = knn.predict(x_test)
acc_knn = round(knn.score(x_train, y_train) * 100, 2)
acc_knn
```

Out[27]:

```
91.33
```

Gussian Naive Bayes

In [28]:

```
# Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
gaussian = GaussianNB()
gaussian.fit(x_train, y_train)
y_pred = gaussian.predict(x_test)
acc_gaussian = round(gaussian.score(x_train, y_train) * 100, 2)
acc_gaussian
```

Out[28]:

```
88.33
```

Perceptron Model/Algorithm analysis

In [29]:

```
# Perceptron
from sklearn.linear_model import Perceptron
perceptron = Perceptron()
perceptron.fit(x_train, y_train)
y_pred = perceptron.predict(x_test)
acc_perceptron = round(perceptron.score(x_train, y_train) * 100, 2)
acc_perceptron
```

Out[29]:

```
76.67
```

Stochastic Gradient Descent

In [30]:

```
# Stochastic Gradient Descent
from sklearn import linear_model
clf = linear_model.SGDClassifier()
#clf = SGDClassifier()
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
acc_sgd = round(clf.score(x_train, y_train) * 100, 2)
acc_sgd
```

Out[30]:

```
82.33
```

Decision Tree Algorithm

In [31]:

```
# Decision Tree

decision_tree = DecisionTreeClassifier()
decision_tree.fit(x_train, y_train)
y_pred = decision_tree.predict(x_test)
```

```
acc_decision_tree = round(decision_tree.score(x_train, y_train) * 100, 2)
acc_decision_tree
```

Out[31]:

```
100.0
```

Random Forest

In [32]:

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(x_train, y_train)
y_pred = random_forest.predict(x_test)
random_forest.score(x_train, y_train)
acc_random_forest = round(random_forest.score(x_train, y_train) * 100, 2)
acc_random_forest
```

Out[32]:

```
100.0
```

Congratulation !!!! Got an Awesome prediction of 100% accuracy by using Random forest and Decision Tree Algorithm

Model evaluation

We can now rank our evaluation of all the models to choose the best one for our problem. While both Decision Tree and Random Forest score the same, we choose to use Random Forest as they correct for decision trees' habit of overfitting to their training set.

In [33]:

```
models = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes', 'Perceptron',
              'Stochastic Gradient Decent', 'Linear SVC',
              'Decision Tree'],
    'Score': [acc_svc, acc_knn, acc_log,
              acc_random_forest, acc_gaussian, acc_perceptron,
              acc_sgd, acc_linear_svc, acc_decision_tree]})
models.sort_values(by='Score', ascending=False)
```

Out[33]:

	Model	Score
3	Random Forest	100.00
8	Decision Tree	100.00
0	Support Vector Machines	91.33
1	KNN	91.33
4	Naive Bayes	88.33
7	Linear SVC	82.67
2	Logistic Regression	82.33
6	Stochastic Gradient Decent	82.33
5	Perceptron	76.67

In [34]:

```
#submission = pd.DataFrame({
    #    "User_ID": x_test["User_ID"],
     #    "Purchased": Y_pred
```

```
    # })
#submission.to_csv('../output/submission.csv')
```