MACHINE LEARNING

(Finding Donors)

Summer Internship Report Submitted in partial fulfillment

of the requirement for undergraduate degree of

Bachelor of Technology

In

Computer Science Engineering

By

Kundla Naveen

221710313027

Under the Guidance of

Assistant Professor

GITAM
(DEEMED TO BE UNIVERSITY)
VISAKHAPATNAM ❁ HYDERABAD ❁ BENGALURU

Department Of Electronics and Communication Engineering
GITAM School of Technology
GITAM (Deemed to be University)
Hyderabad-502329
June 2019

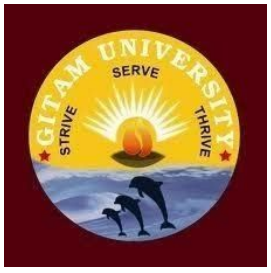
DECLARATION

I submit this industrial training work entitled "Finding Donors" to GITAM (Deemed To Be University), Hyderabad in partial fulfillment of the requirements for the award of the degree of "**Bachelor of Technology**" in "**Computer Science Engineering**". I declare that it was carried out independently by me under the guidance of **Mr.**, Asst. Professor, GITAM (Deemed To Be University), Hyderabad, India.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Place: HYDERABAD — Kundla Naveen

Date: — 221710313027



GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled **"Finding Donors"** is being submitted by Kundla Naveen (221710313027) in partial fulfillment of the requirement for the award of **Bachelor of Technology in Computer Science Engineering** at GITAM (Deemed To Be University), Hyderabad during the academic year 2018-19

It is faithful record work carried out by her at the **Computer Science Engineering Department**, GITAM University Hyderabad Campus under my guidance and supervision.

Mr.
Assistant Professor
Department of CSE

Dr.
Professor and HOD
Department of CSE

ACKNOWLEDGEMENT

Apart from my effort, the success of this internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful competition of this internship.

I would like to thank respected **Dr. N. Siva Prasad,** Pro Vice Chancellor, GITAM Hyderabad and **Dr. CH. Sanjay,** Principal, GITAM Hyderabad

I would like to thank respected **Dr,** Head of the Department of Computer Science Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a internship report. It helped me a lot to realize of what we study for.

I would like to thank the respected faculties **Mr.** who helped me to make this internship a successful accomplishment.

I would also like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Kundla Naveen
221710313027

ABSTRACT

Machine learning algorithms are used to predict the values from the data set by splitting the data set in to train and test and building Machine learning algorithms models of higher accuracy to predict the values is the primary task to be performed on Donors data set.In this project, we will use a number of different supervised algorithms to precisely predict individuals' salary using data collected from the 1994 U.S. Census.

We will then choose the best candidate algorithm from preliminary results and further optimize this algorithm to best model the data. Our goal with this implementation is to build a model that accurately predicts whether an individual makes more than $50,000.

This sort of task can arise in a non-profit setting, where organizations survive on donations. Understanding an individual's income can help a non-profit better understand how large of a donation to request, or whether or not they should reach out to begin with. As from our previous research we have found out that the individuals who are most likely to donate money to a charity are the ones that make more than $50,000.

Table of Contents: Page No.

TABLE OF FIGURES **Page No.**

CHAPTER 1

MACHINE LEARNING

1.1 INTRODUCTION:

Machine Learning(ML) is the scientific study of algorithms and statistical models that computer systems use in order to perform a specific task effectively without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of Artificial Intelligence(AI).

1.2 IMPORTANCE OF MACHINE LEARNING:

Consider some of the instances where machine learning is applied: the self-driving Google car, cyber fraud detection, online recommendation engines—like friend suggestions on Facebook, Netflix showcasing the movies and shows you might like, and "more items to consider" and "get yourself a little something" on Amazon—are all examples of applied machine learning. All these examples echo the vital role machine learning has begun to take in today's data-rich world.

Machines can aid in filtering useful pieces of information that help in major advancements, & we are already seeing how this technology is being implemented in a wide variety of industries.

With the constant evolution of the field, there has been a subsequent rise in the uses, demands, and importance of machine learning. Big data has become quite a buzzword in the last few years; that's in part due to increased sophistication of machine learning, which helps analyze those big chunks of big data. Machine learning has also changed the way data extraction, and interpretation is done by involving automatic sets of generic methods that have replaced traditional statistical technique.

The process flow depicted here represents how machine learning works

Figure 1 : The Process Flow

1.3 USES OF MACHINE LEARNING:

Earlier in this article, we mentioned some applications of machine learning. To understand the concept of machine learning better, let's consider some more examples: web search results, real-time ads on web pages and mobile devices, email spam filtering, network intrusion detection, and pattern and image recognition. All these are by-products of applying machine learning to analyze huge volumes of data

Traditionally, data analysis was always being characterized by trial and error, an approach that becomes impossible when data sets are large and heterogeneous. Machine learning comes as the solution to all this chaos by proposing clever alternatives to analyzing huge volumes of data.

By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning can produce accurate results and analysis.

1.4 TYPES OF LEARNING ALGORITHMS:

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

1.4.1 Supervised Learning :

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning.

Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to "learn" how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data.

Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign.

Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification.

1.4.2 Unsupervised Learning:

When an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

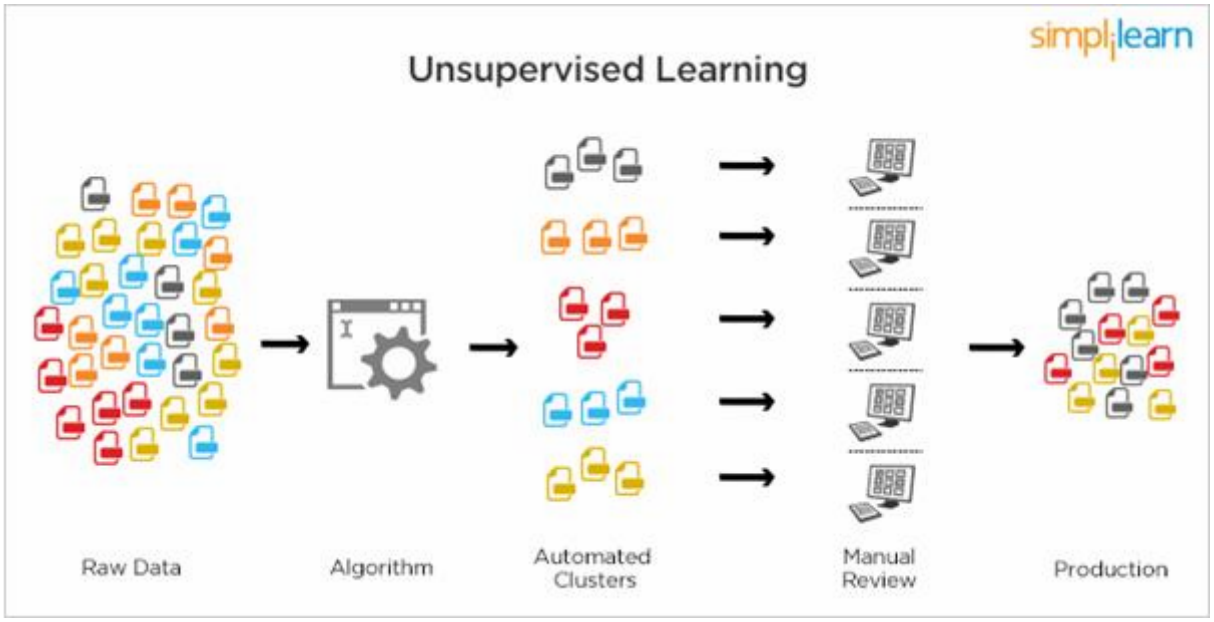


Figure 2 : Unsupervised Learning

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbor mapping, singular value decomposition, and k-means clustering. Basically, online recommendations, identification of data outliers, and segment text topics are all examples of unsupervised learning.

1.4.3 Semi Supervised Learning:

As the name suggests, semi-supervised learning is a bit of both supervised and unsupervised learning and uses both labeled and unlabeled data for training. In a typical scenario, the algorithm would use a small amount of labeled data with a large amount of unlabeled data.

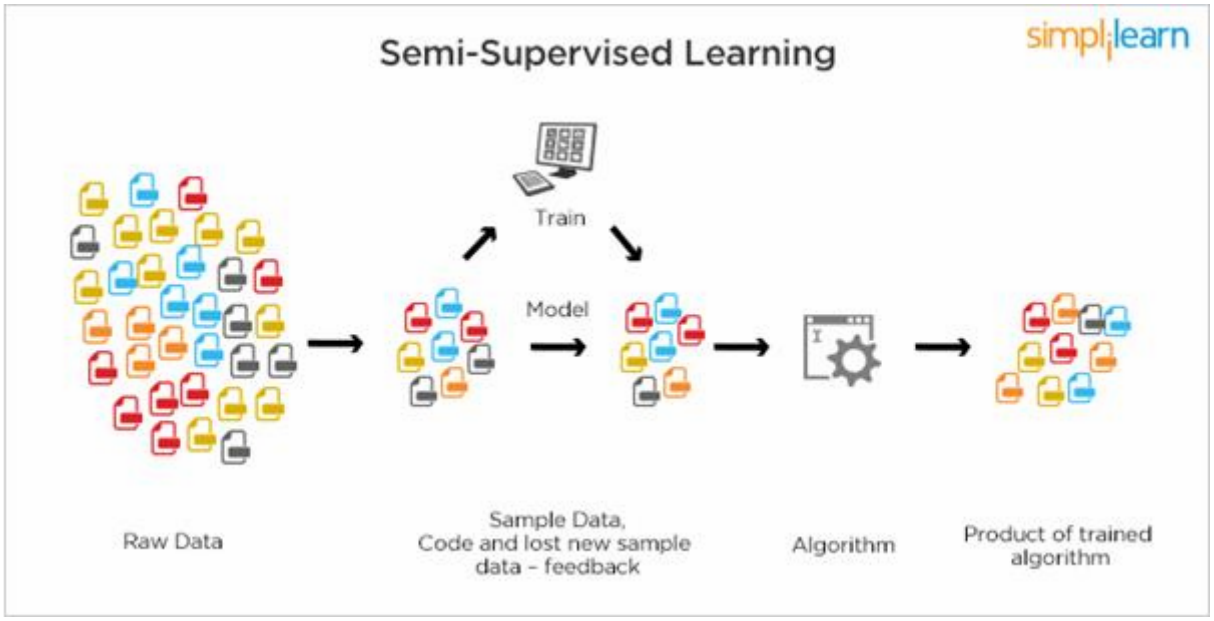


Figure 3 : Semi Supervised Learning

1.5 RELATION BETWEEN DATA MINING,MACHINE LEARNING AND DEEP LEARNING:

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovers previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions.

Deep learning, on the other hand, uses advanced computing power and special

types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

CHAPTER 2

PYTHON

Basic programming language used for machine learning is : PYTHON

2.1 INTRODUCTION TO PYTHON:

- Python is a high-level, interpreted, interactive and object-oriented scripting language.
- Python is a general purpose programming language that is often applied in scripting roles
- Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.
- Python is Interactive: You can sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: Python supports the Object-Oriented style or technique of programming that encapsulates code within objects.

2.2 HISTORY OF PYTHON:

- Python was developed by GUIDO VAN ROSSUM in early 1990's
- Its latest version is 3.7 , it is generally called as python3

2.3 FEATURES OF PYTHON:

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax, This allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintaining.
- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases: Python provides interfaces to all major commercial databases.
- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

2.4 HOW TO SETUP PYTHON:

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

2.4.1 Installation(using python IDLE):

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.
- Download python from www.python.org
- When the download is completed, double click the file and follow the instructions to install it.
- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.

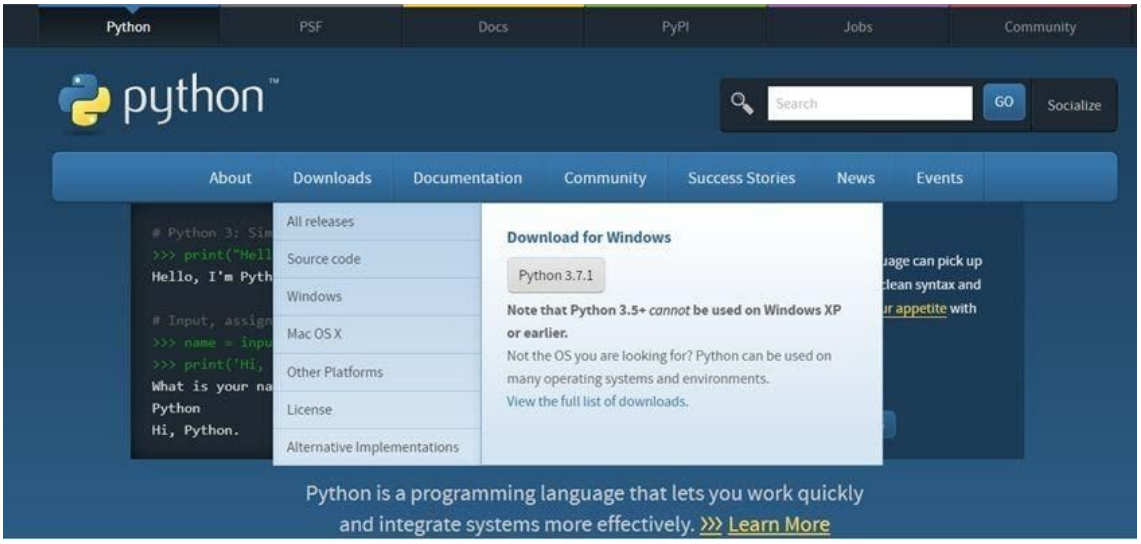


Figure 4 : Python download

2.4.2 Installation(using Anaconda):

- Python programs are also executed using Anaconda.
- Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.
- Conda is a package manager quickly installs and manages packages.

- In WINDOWS:
- In windows
    - Step 1: Open Anaconda.com/downloads in web browser.
    - Step 2: Download python 3.4 version for (32-bitgraphic installer/64 -bit graphic installer)
    - Step 3: select installation type(all users)
    - Step 4: Select path(i.e. add anaconda to path & register anaconda as default python 3.4) next click install and next click finish
    - Step 5: Open jupyter notebook (it opens in default browser)

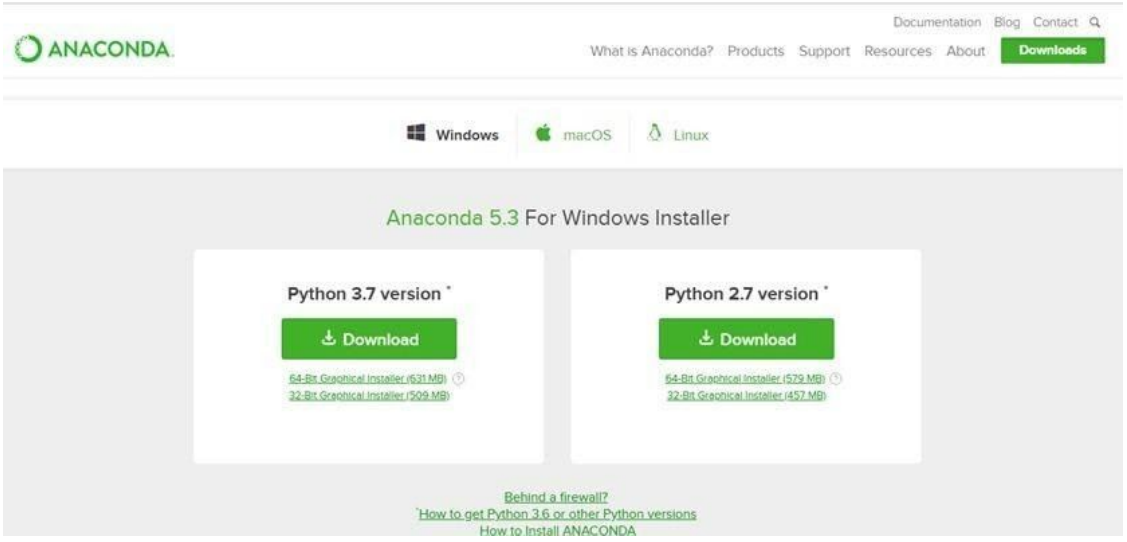


Figure 5 : Anaconda download

Figure 6 : Jupyter notebook

2.5 PYTHON VARIABLE TYPES:

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Variables are nothing but reserved memory locations to store values.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types –
    - Numbers
    - Strings
    - Lists

- Tuples
- Dictionary

2.5.1 Python Numbers:

- Number data types store numeric values. Number objects are created when you assign a value to them.
- Python supports four different numerical types − int (signed integers) long (long integers, they can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

2.5.2 Python Strings:

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

2.5.3 Python Lists:

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets

([]).

- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

2.5.4 Python Tuples:

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated.
- Tuples can be thought of as read-only lists.
- For example − Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add elements to a tuple. Tuples have no append or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

2.5.5 Python Dictionary:

- Python's dictionaries are kind of hash table type. They work like associative arrays

or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).
- You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of a list.
- What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

2.6 PYTHON FUNCTION:

2.6.1 Defining a Function:

You can define functions to provide the required functionality. Here are simple rules to define a function in Python. Function blocks begin with the keyword def followed by the function name and parentheses (i.e.()).

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses

The code block within every function starts with a colon (:) and is indented. The statement returns [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

2.6.2 Calling a Function:

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

2.7 PYTHON USING OOP's CONCEPTS:

2.7.1 Class:

- Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- Data member: A class variable or instance variable that holds data associated with a class and its objects.
- Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.

- Defining a Class:
    - We define a class in a very similar way how we define a function.
    - Just like a function ,we use parentheses and a colon after the class name(i.e. ():) when we define a class. Similarly, the body of our class is

indented like a functions body is.

```
def my_function():
    # the details of the
    # function go here
```

```
class MyClass():
    # the details of the
    # class go here
```

Figure 7 : Defining a Class

2.7.2 init method in Class:

- The init method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.
- The init method has a special name that starts and ends with two underscores: init ().

CHAPTER 3 CASE STUDY

3.1 PROBLEM STATEMENT:

Finding donors using Machine Learning.

3.2 DATA SET:

1. age
2. workclass
3. fnlwgt
4. education
5. education-num
6. marital status
7. occupation
8. relationship
9. race
10. sex
11. capital-gain
12. capital-loss
13. hours-per-week
14. native-country
15. salary

3.3 OBJECTIVE OF THE CASE STUDY:

CharityML is a fictitious charity organization that wants to expand their potential donor base by sending letters to residents of the region where it is located, but only to the ones who would most likely donate. After nearly 32000 letters were sent to people in the community, CharityML determined that every donation they received came from someone that was making more than $50000 annually. So our goal is to build an algorithm to best identify potential donors and reduce overhead cost of sending mail.

CHAPTER 4

MODEL BUILDING

4.1 Data Collection:

Preprocessing of the data actually involves the following steps:

4.1.1 Getting the data set:

We can get the data set from the database.

4.1.2 Importing the libraries:

We have to import the libraries as per the requirement of the algorithm.

We are using a python file(visuals.py) for visualisation. I have taken it as Library .Because it is not inbuilt , it is a user defined.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from IPython.display import display
import visuals as vs
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import GridSearchCV
```

fig 8. importing libraries

4.1.3 Importing the data-set:

We need to load data the data .The most common format for data loading is CSV(comma-seperated-values).

```
data = pd.read_csv ("donars.csv")
data
```

]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	salary
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	United-States	<=50K
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	United-States	>50K
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	United-States	<=50K
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40	United-States	>50K

32561 rows × 15 columns

fig 9.reading the data

4.2 Exploratory Data analysis:

4.2.1 Column names

```
In [4]: list(data.columns)

Out[4]: ['age',
         'workclass',
         'fnlwgt',
         'education',
         'education-num',
         'marital-status',
         'occupation',
         'relationship',
         'race',
         'sex',
         'capital-gain',
         'capital-loss',
         'hours-per-week',
         'native-country',
         'salary']
```

fig 10.total no of columns

4.2.2 Statistical summary data

```
data.describe()
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

fig 11.statistical data

4.2.3 Summary of data frame

```
data.info()
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64
 3   education       32561 non-null  object
 4   education-num   32561 non-null  int64
 5   marital-status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital-gain    32561 non-null  int64
 11  capital-loss    32561 non-null  int64
 12  hours-per-week  32561 non-null  int64
 13  native-country  32561 non-null  object
 14  salary          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

fig12 .Display all information

Total no of records

```
In [8]: n_of_records = len(data)
        n_of_records

Out[8]: 32561

In [9]: df = data.groupby(['salary']).salary.count()

In [10]: df

Out[10]: salary
          <=50K    24720
          >50K      7841
         Name: salary, dtype: int64

In [11]: ##Income of people greater than 50K
         n_greater_50k = df[1]
         n_greater_50k

Out[11]: 7841

In [12]: ##Income of people atmost 50k
         n_at_most_50k = df[0]
         n_at_most_50k

Out[12]: 24720
```

fig13. segregating the data

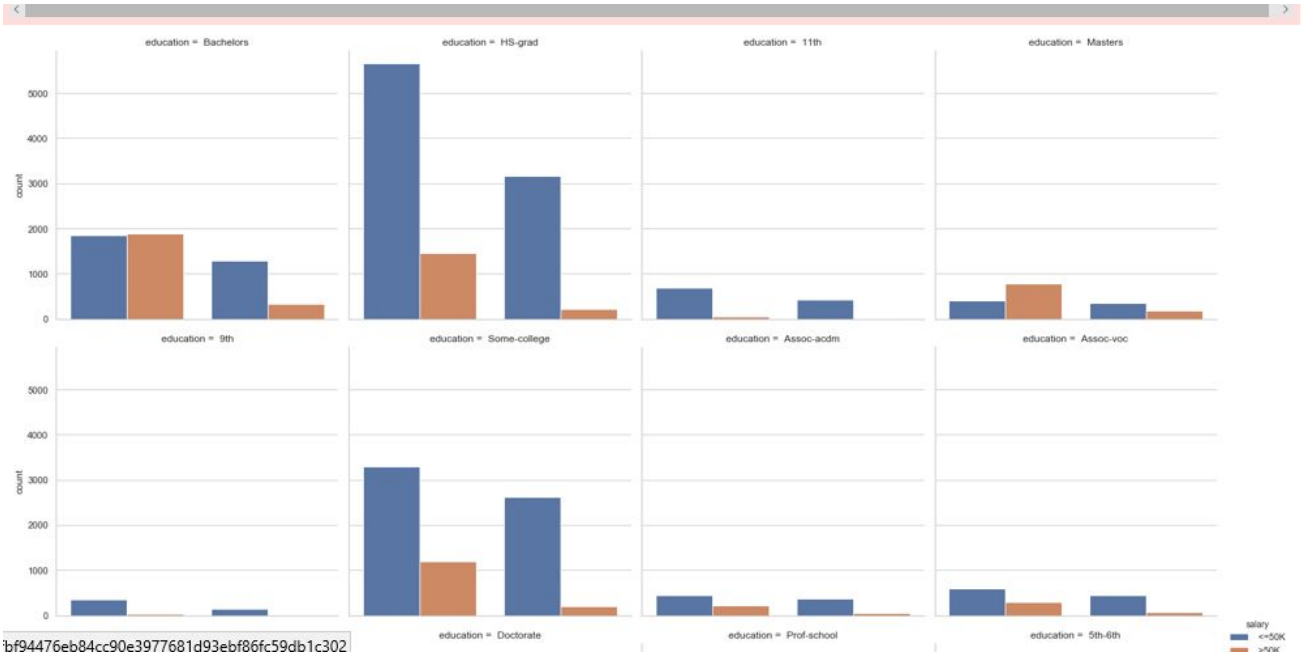
Here,the total length of the data is 32561.By using groupby() we just got to know that our targeted column i.e.,Salary has two outputs , the one who earns more than 50k and less than or equal to 50k.

4.2.5 Preparing the Data:

- Data must be preprocessed in order to be used in Machine Learning algorithms. This preprocessing phase includes the cleaning, formatting and restructuring of the data.
- For this dataset there aren't empty entries nor invalid ones, but,there are some features that must be adjusted. This task will dramatically improve the results and the predictive power of our model.
- We can also visualize the relationship between different features of an individual, and their incomes.Let's see breakdown of the counts of people earning above or below 50K based on their sex and education levels.

```
sns.set(style="whitegrid", color_codes=True)
sns.factorplot("sex", col='education', data=data, hue='salary', kind="count", col_wrap=4);
```

```
C:\Users\home\anaconda3\lib\site-packages\seaborn\categorical.py:3669: UserWarning: The `factorplot` function has bee
n renamed to `catplot`. The original name will be removed in a future release. Please update your code. Note that the
default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`.
  warnings.warn(msg)
```



bf94476eb84cc90e3977681d93ebf86fc59db1c302

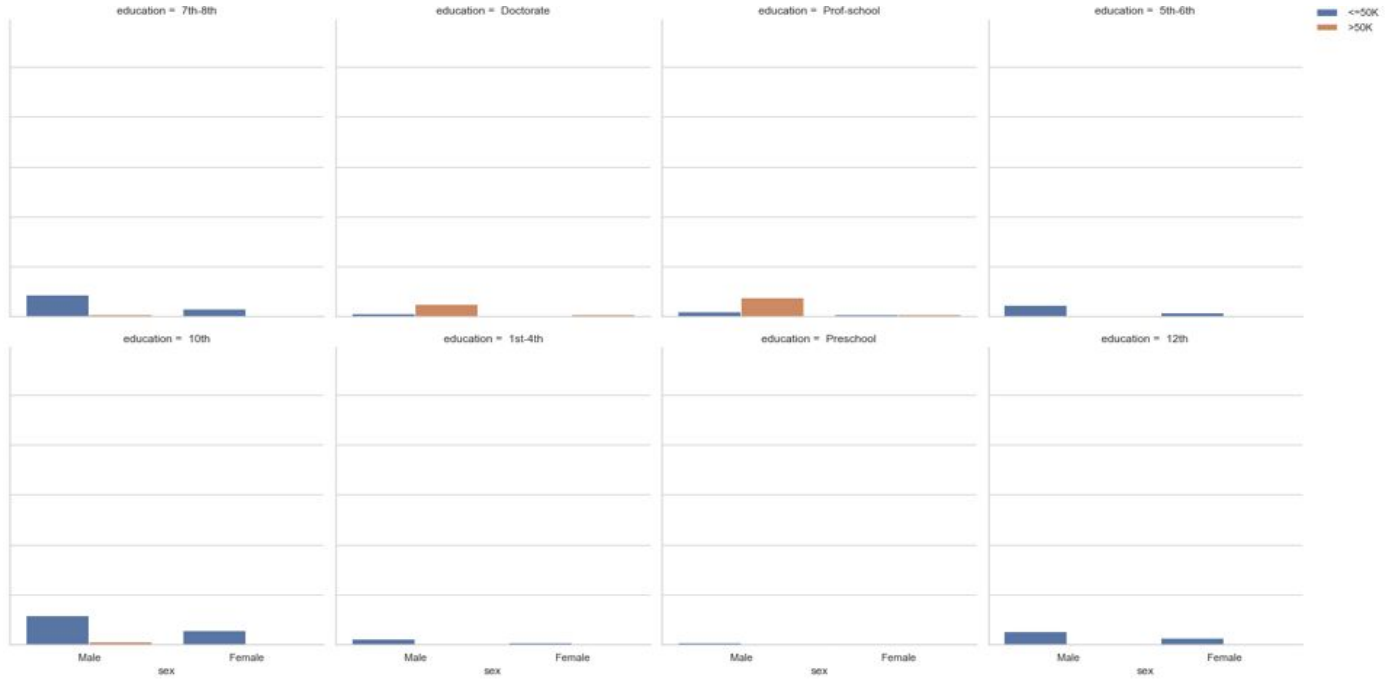


fig14 .statistical graph plotting

4.2.5.1 Transforming Skewed Continuous Features:

A dataset may sometimes contain at least one feature whose values tend to lie near a single number, but will also have a non-trivial number of vastly larger or smaller values than that single number. Algorithms can be sensitive to such distributions of values and can underperform if the range is not properly normalized. With the census dataset two features fit this description: 'capital-gain' and 'capital-loss'.

Let's plot a histogram of these two features and see how they are distributed.

```
income_raw = data['salary']
features_raw = data.drop('salary', axis = 1)

# Visualize skewed continuous features of original data
vs.distribution(data)
```

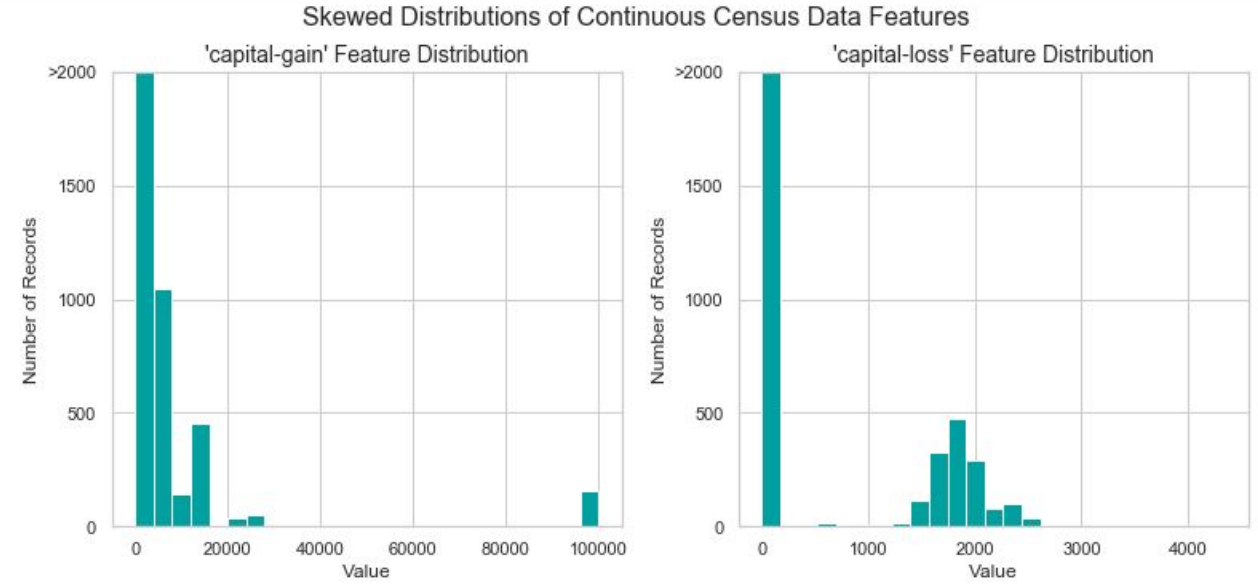


fig15. Histogram for skewed distribution

For highly-skewed feature distributions such as 'capital-gain' and 'capital-loss', it is common practice to apply a logarithmic transformation on the data so that the very large and very small values do not negatively affect the performance of a learning algorithm. Using a logarithmic transformation significantly reduces the range of values caused by outliers.

```
# Log-transform the skewed features
skewed = ['capital-gain', 'capital-loss']
features_raw[skewed] = data[skewed].apply(lambda x: np.log(x + 1)) #add 1

# Visualize the new log distributions
vs.distribution(features_raw, transformed = True)
```

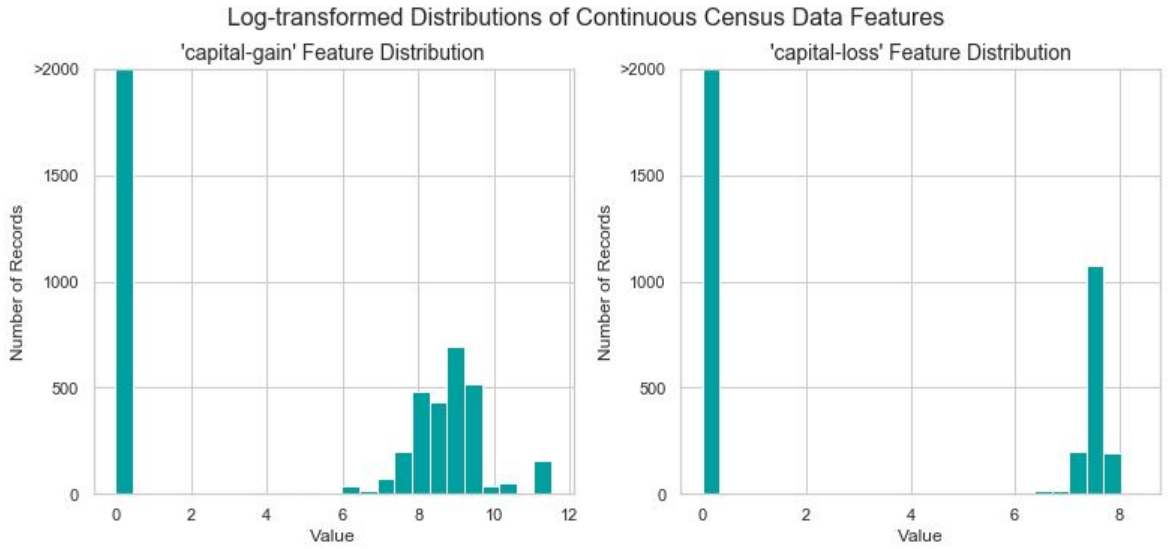


fig16. Histogram for Log-transformed distribution

4.3 Data Preprocessing:

From the table in Exploring the Data above, we can see there are several features for each record that are non-numeric. Typically, learning algorithms expect input to be numeric, which requires that non-numeric features (called *categorical variables*) be converted. One popular way to convert categorical variables is by using the one-hot encoding scheme. One-hot encoding creates a *"dummy"* variable for each possible category of each non-numeric feature.

ONEHOTENCODER

[18]:
```
#Creating dummies
features = pd.get_dummies(features_raw)


from sklearn.preprocessing import LabelEncoder,OneHotEncoder
labelencoder = LabelEncoder()
data['salary'] = labelencoder.fit_transform(data['salary'])
labelencoder = LabelEncoder()
data['sex'] = labelencoder.fit_transform(data['sex'])
labelencoder = LabelEncoder()
data['race'] = labelencoder.fit_transform(data['race'])

data
```

18]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	salary
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	4	1	2174	0	40	United-States	0
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	4	1	0	0	13	United-States	0
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	4	1	0	0	40	United-States	0
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	2	1	0	0	40	United-States	0
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	2	0	0	0	40	Cuba	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	4	0	0	0	38	United-States	0
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	4	1	0	0	40	United-States	1
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	4	0	0	0	40	United-States	0
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	4	1	0	0	20	United-States	0
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	4	0	15024	0	40	United-States	1

32561 rows × 15 columns

fig17.OneHotEncoder

4.3.2 Normalizing Numerical Features:

In addition to performing transformations on features that are highly skewed, it is often good practice to perform some type of scaling on numerical features. Applying a scaling to the data does not change the shape of each feature's distribution (such as 'capital-gain' or 'capital-loss' above); however, normalization ensures that each feature is treated equally when applying supervised learners. Note that once scaling is applied, observing the data in its raw form will no longer have the same original meaning, as example below.

Scaling

Normalizing Numerical Features:

```
: from sklearn.preprocessing import MinMaxScaler

# Initialize a scaler, then apply it to the features
scaler = MinMaxScaler()
numerical = ['age', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']
features_raw[numerical] = scaler.fit_transform(data[numerical])

# Show an example of a record with scaling applied
display(features_raw.head(5))
```

```
# Show an example of a record with scaling applied
display(features_raw.head(5))
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country
0	0.301370	State-gov	77516	Bachelors	0.800000	Never-married	Adm-clerical	Not-in-family	White	Male	0.02174	0.0	0.397959	United-States
1	0.452055	Self-emp-not-inc	83311	Bachelors	0.800000	Married-civ-spouse	Exec-managerial	Husband	White	Male	0.00000	0.0	0.122449	United-States
2	0.287671	Private	215646	HS-grad	0.533333	Divorced	Handlers-cleaners	Not-in-family	White	Male	0.00000	0.0	0.397959	United-States
3	0.493151	Private	234721	11th	0.400000	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0.00000	0.0	0.397959	United-States
4	0.150685	Private	338409	Bachelors	0.800000	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0.00000	0.0	0.397959	Cuba

fig18. Scaling using MinMaxScalar

4.3.3 Pick a Model/Algorithm:

4.3.3.1 Gaussian Naive Bayes :

- The strengths of the model are: It is simple and fast classifier that provides good results with little tunning of the model's hyperparameters. In addition, it does not require a large amount of data to be properly trained.
- The **weaknesses** of the model are: It has a strong feature independence assumptions. If we do not have occurrence of a class label and a certain attribute value together (e.g. class = 'nice', shape =

'sphere') then the frequency-based probability estimate will be zero, so given the conditional independence assumption, when all the probabilities are multiplied we will get zero, which will affect the posterior probability estimate.

- One possible real world application where this model can be applied is for text learning.
- It is a good candidate because it is an efficient model and can deal with many features (the data set contains 98 features).

4.3.3.2 Random Forests :

- The strengths of the model are: It works well with binary features, as it is an ensembling of decision trees. It does not expect linear features. It works well with high dimensional spaces and large number of training examples.
- The main weakn**ess** is that it may overfit when dealing with noisy data.
- One possible real world application where this model can be applied is for predicting stock market prices.
- It is a good candidate because it is often a quite accurate classificator and works well with binary features and high dimensional datasets.

4.3.3.3 Logistic Regression:

- Logistic Regression is used when the dependent variable(target) is categorical.
- Logistic Regression is the appropriate regression analysis to conduct when the dependent variable is

dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis and it predicts the probability.

*Example: Yes or No, get a disease or not, pass or fail, defective or non-defective, etc.,

- Also called a classification algorithm, because we are classifying the data. It predicts the probability associated with each dependent variable category.

CHAPTER 5
TRAINING THE MODEL:

5.1 Method 1:

- Splitting the data : after the preprocessing is done then the data is split into train and test sets
- In Machine Learning in order to access the performance of the classifier. You train the classifier using 'training set' and then test the performance of your classifier on unseen 'test set'. An important point to note is that during training the classifier only uses the training set . The test set must not be used during training the classifier. The test set will only be available during testing the classifier.
- training set - a subset to train a model.(Model learns patterns between Input and Output)
- test set - a subset to test the trained model.(To test whether the model has correctly learnt)
- The amount or percentage of Splitting can be taken as specified (i.e. train data = 75% , test data =25% or train data = 80% , test data= 20%)

- First we need to identify the input and output variables and we need to separate the input set and output set
- In scikit learn library we have a package called model_selection in which train_test_split method is available .we need to import this method
- This method splits the input and output data to train and test based on the percentage specified by the user and assigns them to four different variables(we need to mention the variables)

train_test_split

```
]: # Import train_test_split
from sklearn.model_selection import train_test_split

# Split the 'features' and 'income' data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features,
                                                    income_raw,
                                                    test_size = 0.2,
                                                    random_state = 0)

# Showing the results of the split
print(X_train.shape[0])
print(X_test.shape[0])
print(y_train.shape[0])
print(y_test.shape[0])
```

```
26048
6513
26048
6513
```

fig:19.Splitting the Data

CHAPTER 6
MODEL BUILDING AND EVALUATION:

6.1 Naive Bayes

Naive Bayes

```
# Apply the naive Bayes Algorithm
# Import BernNB
from sklearn.naive_bayes import BernoulliNB
# creating an object for BerNB
model_BernNB = BernoulliNB()
```

fig20. Importing BerNB

- Here we are importing BernoulliNavieBase from skearn.naive_bayes and we are creating an object for BerNB.

```
# Applying the algorithm to the data
# objectName.fit(Input,Output)
model_BernNB.fit(X_train, y_train)
```

```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
```

fig21. Applying algorithm

- -Applying the algorithm to the data

```
]: y_train_pred = model_BernNB.predict(X_train)
```

```
]: # compare the actual values(y_test) with predicted values(y_test_pred)
from sklearn.metrics import confusion_matrix,classification_report
confusion_matrix(y_train,y_train_pred)
```

```
]: array([[14826,  4976],
       [ 1297,  4949]], dtype=int64)
```

fig22. Comparing actual values with predicted

- comparing the actual values(y_test) with predicted values(y_test_pred)

```
]: print(classification_report(y_train,y_train_pred))
```

```
              precision    recall  f1-score   support

       <=50K       0.92      0.75      0.83     19802
        >50K       0.50      0.79      0.61      6246

    accuracy                           0.76     26048
   macro avg       0.71      0.77      0.72     26048
weighted avg       0.82      0.76      0.77     26048
```

fig:23.Classification report for training data

- -classification report for training data

```
from sklearn.metrics import accuracy_score
accuracy_score(y_train,y_train_pred)
```

```
0.7591753685503686
```

fig:24.Accuracy_score

- -here we are checking the accuracy score of training data

```
50]: # Applying the algorithm to the data
     # objectName.fit(Input,Output)
     model_BernNB.fit(X_test, y_test)

50]: BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)

51]: y_test_pred = model_BernNB.predict(X_test)

52]: # compare the actual values(y_test) with predicted values(y_test_pred)
     from sklearn.metrics import confusion_matrix,classification_report
     confusion_matrix(y_test,y_test_pred)

52]: array([[3692, 1226],
            [ 350, 1245]], dtype=int64)
```

fig:25.confusion Matrix for testing data

- Here we are applying algorithm tho the data i.e, testing data and we are comparing the actual values(y_test) with predicted values(y_test_pred)

```
print(classification_report(y_test,y_test_pred))
```

	precision	recall	f1-score	support
<=50K	0.91	0.75	0.82	4918
>50K	0.50	0.78	0.61	1595
accuracy			0.76	6513
macro avg	0.71	0.77	0.72	6513
weighted avg	0.81	0.76	0.77	6513

fig26.Classification report for testing data

```
: from sklearn.metrics import accuracy_score
  accuracy_score(y_test,y_test_pred)

: 0.7580224167050514

: model_BernNB_score = accuracy_score(y_test,y_test_pred)*100

: model_BernNB_score

: 75.80224167050514
```

fig 27:accuracy score for Naive bayes

- classification report and accuracy score of testing data.

6.2 Random Forest Classifier

```
#import,inttialize and fit
#import the RFC From sklearn
from sklearn.ensemble import RandomForestClassifier

#intialize the object for RFC
rfc = RandomForestClassifier(n_estimators = 40)

#fit RFC to the dataset
rfc.fit(X train, y train)
```

fig:28.Importing RandomForestClassifier

-Here we are importing RandomForestClassifer from sklearn.ensemble.and we are intializing an object for random forest classifier.

```
#predict on training data
#syntax:objectname.predict(Inputvalues)
y_pred_train = rfc.predict(X_train)

from sklearn.metrics import confusion_matrix,classification_report
print(classification_report(y_train, y_pred_train))
```

```
              precision    recall  f1-score   support

       <=50K       1.00      1.00      1.00     19802
        >50K       1.00      1.00      1.00      6246

    accuracy                           1.00     26048
   macro avg       1.00      1.00      1.00     26048
weighted avg       1.00      1.00      1.00     26048
```

fig:29.Classification report for training data

- -Here we are doing prediction for the training data and we have taken classification report from sklearn.metrics.

```
from sklearn.metrics import accuracy_score
accuracy_score(y_train,y_pred_train)
```

```
0.9990402334152334
```

fig:30.Accuracy score for training data

- Here it's the accuracy score for the training data.

```
#predicition on test data(unseen data)
y_pred_test = rfc.predict(X_test)
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

       <=50K       0.88      0.92      0.90      4918
        >50K       0.72      0.61      0.66      1595

    accuracy                           0.85      6513
   macro avg       0.80      0.77      0.78      6513
weighted avg       0.84      0.85      0.84      6513
```

fig:31.Classification report for testing data

- Here it's the prediction of the testing data.and we have also taken the Classification report.

```
]: accuracy_score(y_test,y_pred_test)

]: 0.8472286196837095

]: #crossvalidationscore
```

```
rfc_score = accuracy_score(y_test,y_pred_test)*100

rfc_score

84.73821587594043
```

fig:32.Accuracy score for testing data

-we have also got the Accuracy score of the testing data.

6.3 Logistic Regression

```
from sklearn.linear_model import LogisticRegression
logr=LogisticRegression()

logr.fit(X_train,y_train) #train

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)

y_train_pred=logr.predict(X_train)
```

fig:33. Importing Logistic Regression and fitting data

-Here we have imported LogisticRegression from sklearn.linear_model and we are initializing an object for Logistic regression i.e.., logr

```
from sklearn.metrics import confusion_matrix,classification_report
confusion_matrix(y_train,y_train_pred)
```

```
array([[19802,     0],
       [ 6246,     0]], dtype=int64)
```

```
print(classification_report(y_train,y_train_pred))
```

```
C:\Users\home\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precision
and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

	precision	recall	f1-score	support
<=50K	0.76	1.00	0.86	19802
>50K	0.00	0.00	0.00	6246
accuracy			0.76	26048
macro avg	0.38	0.50	0.43	26048
weighted avg	0.58	0.76	0.66	26048

fig:34.Confusion matrix and classification report of training data

-here we have taken confusion matrices for training data and classification reports for training data.

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_test_pred)
```

```
0.7551051742668509
```

```
logr_score = accuracy_score(y_test,y_test_pred)*100
```

```
logr_score
```

```
75.80224167050514
```

fig:35.Accuracy score of training data

- we have got the accuracy score for the training data.

```
logr.fit(X_test,y_test)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
y_test_pred = logr.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix,classification_report
confusion_matrix(y_test,y_test_pred)
```

```
array([[4918,    0],
       [1595,    0]], dtype=int64)
```

fig:36.Confusion matrix for testing data

- -Now we have taken the testing data,and taken the confusion matrixing for the training data

```
print(classification_report(y_test,y_test_pred))
```

```
              precision    recall  f1-score   support

       <=50K       0.76      1.00      0.86      4918
        >50K       0.00      0.00      0.00      1595

    accuracy                           0.76      6513
   macro avg       0.38      0.50      0.43      6513
weighted avg       0.57      0.76      0.65      6513

C:\Users\home\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precision
and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

fig:37.Classification report for testing data

- -It’s the classification report for testing data.

```
: from sklearn.metrics import accuracy_score
  accuracy_score(y_test,y_test_pred)
```

```
: 0.7551051742668509
```

```
logr_score = accuracy_score(y_test,y_test_pred)*100
```

```
logr_score
```

```
75.80224167050514
```

fig:38.Accuracy score

- -we have got the accuracy_score.

Comparison:

By comparing all the three algorithms .I got more accuracy and precision value in RandomForestClassifier.so,It's the best method for the project "FINDING DONORS".

```
# comparing scores
Methods = ['Random Forest','Logistic Regression','Navie Bayes']
scores = np.array([rfc_score, logr_score, model_BernNB_score])
fig, ax = plt.subplots(figsize=(8,6))
sns.barplot(Methods,scores)
plt.title('Prediction')
plt.ylabel('Accuracy')
```

```
Text(0, 0.5, 'Accuracy')
```

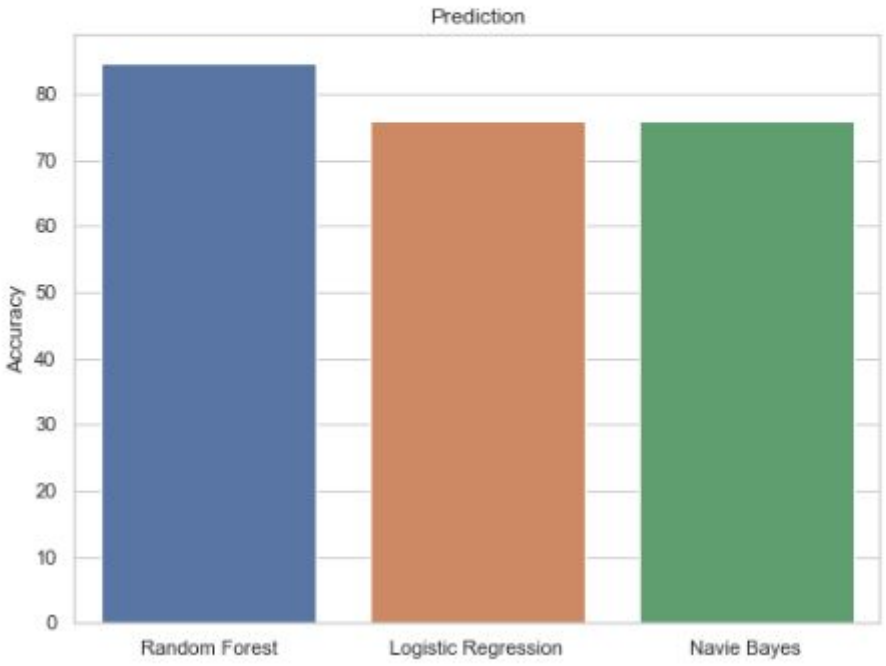


FIG. 39 .Comparing Algorithms

CHAPTER 7: HYPERPARAMETER TUNING

A hyperparameter is a parameter whose value is set before the learning process begins.

Hyperparameter tuning is also tricky in the sense that there is no direct way to calculate how a change in the hyperparameter value will reduce the loss of your model, so we usually resort to experimentation. This starts with us specifying a range of possible values for all the hyperparameters. Now, this is where most get stuck, what values you are going to try, and to answer that question, you first need to understand what these hyperparameters mean and how changing a hyperparameter will affect your model architecture, thereby try to understand how your model performance might change.

The next step after you define the range of values is to use a hyperparameter tuning method, there's a bunch, the most common and expensive being Grid Search

Grid Search CV:

Grid search is a traditional way to perform hyperparameter optimization. It works by searching exhaustively through a specified subset of hyperparameters.

Grid search is the process of performing hyper parameter tuning in order to determine the optimal values for a given model. This is significant as the performance of the entire model is based on the hyper parameter values specified.

Using sklearn's GridSearchCV, we first define our grid of parameters to search over and then run the grid search.

- Hyperparameter tuning for Random Forest Classifier

```
#Import the GridSearchCV
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(rfc, param_grid=param_grid)
grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=None, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=40, n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [3, 5], 'max_leaf_nodes': [20, 40],
                         'min_samples_leaf': [5, 10, 20],
                         'min_samples_split': [15, 20],
                         'min_weight_fraction_leaf': [0.1],
                         'n_estimators': [10, 18, 22]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
grid_search.best_params_
```

```
{'max_depth': 5,
 'max_leaf_nodes': 20,
 'min_samples_leaf': 20,
 'min_samples_split': 15,
 'min_weight_fraction_leaf': 0.1,
 'n_estimators': 10}
```

fig:40 Hyperparameter tuning for Random Forest Classifier

```
rfc = RandomForestClassifier(max_depth= 3,
 max_leaf_nodes=20,
 min_samples_leaf=5,
 min_samples_split=15,
 min_weight_fraction_leaf= 0.1,
 n_estimators=10)
```

```
# We need to fit the model to the data
rfc.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=3, max_features='auto',
                       max_leaf_nodes=20, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=5, min_samples_split=15,
                       min_weight_fraction_leaf=0.1, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

```
# Prediction on test data
pred_test = rfc.predict(X_test)
```

```
#Classification Report of actual values and predicted value(GridSearch)
print(classification_report(y_test, pred_test))
```

```
              precision    recall  f1-score   support

       <=50K       0.76      1.00      0.86      4918
        >50K       0.00      0.00      0.00      1595

    accuracy                           0.76      6513
   macro avg       0.38      0.50      0.43      6513
weighted avg       0.57      0.76      0.65      6513
```

fig 41: Classification report

```
rfc_score=(rfc.score(X_test, pred_test))*100
rfc_score
```

```
100.0
```

fig 42: accuracy score(after grid search)

```
# comparing scores
Methods = ['Random Forest','Logistic Regression','Navie Bayes']
scores = np.array([rfc_score, logr_score, model_BernNB_score])
fig, ax = plt.subplots(figsize=(8,6))
sns.barplot(Methods,scores)
plt.title('Prediction')
plt.ylabel('Accuracy')
```

```
Text(0, 0.5, 'Accuracy')
```

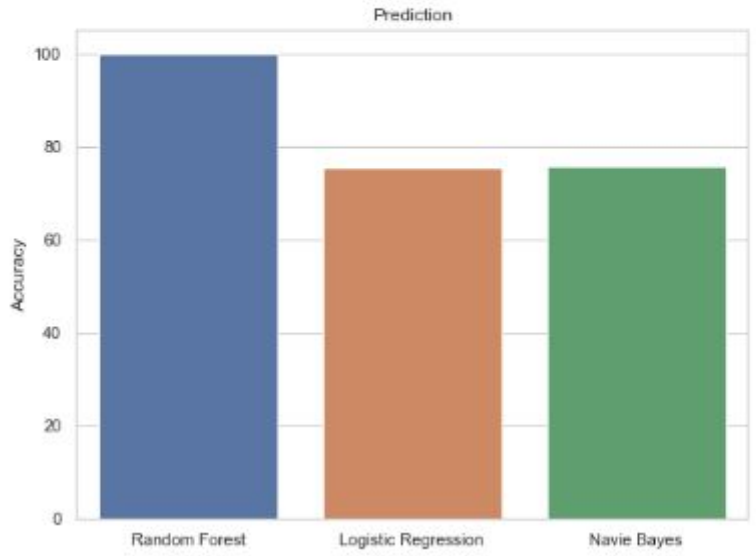


fig 43: comparing score(after grid search)

CONCLUSION:

It is concluded after performing thorough Exploratory Data analysis which include Stats models which are computed to get accuracy and also Heat maps which are computed to get a clear understanding of the data set (which parameter has most abundant effect on the study case) . The RandomForestClassifier algorithm should be Used.

REFERENCES:

[1] https://en.wikipedia.org/wiki/Machine_learning

[2] https://towardsdatascience.com/classification-project-finding-donors-853db66fbb8c