MACHINE LEARNING

(Classify Real or Fake Job Posting)

Summer Internship Report Submitted in partial fulfillment

of the requirement for undergraduate degree of

Bachelor of Technology

In

Computer Science Engineering

By

Kommera Shresta

221710313024

Under the Guidance of

Mr.

Assistant Professor

Department Of Computer Science Engineering GITAM
School of Technology
GITAM (Deemed to be University)
Hyderabad-502329
June 2019

DECLARATION

I submit this industrial training work entitled **"**Classify Real or Fake Job Posting" 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 Kommera Shresta

Date: 221710313024

GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled "**Classify Real or Fake Job Posting**" is being submitted by Kommera Shresta (221710313024) 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 2020-21

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	Professor and HOD
Department of CSE	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.

Kommera Shresta

221710313024

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 classifying real or fake job posting.

To get a better understanding and work on a strategical approach for finding fake job posting, I have adapted the view point of looking at fraudulent and for further deep understanding of the problem, I have taken title, description, company profile, requirements, telecommuting, has company logo, has questions, employment type, Required experience, required education, industry ,city , country name, and my primary objective of this case study was to look up the factors which were to avoid fraudulent post for job .

Different classifiers are used for checking fraudulent post in the web and the results of those classifiers are compared for identifying the best employment fraud detection model.

Table of Contents:

LIST OF FIGURES

LIST of Figures

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, and 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 techniques.

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.

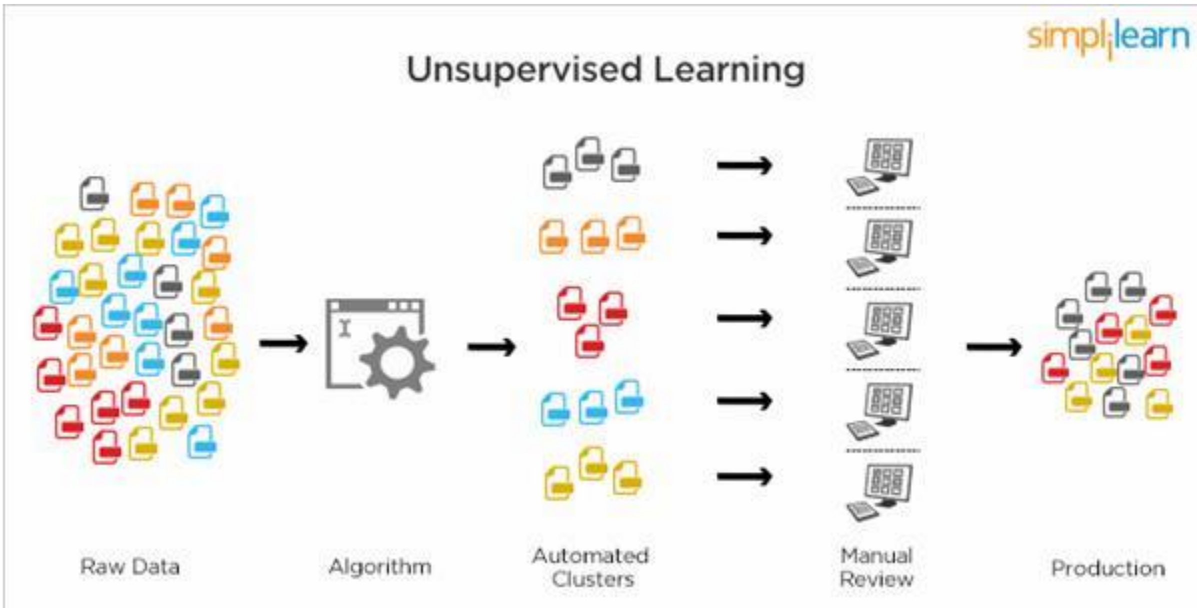


Figure 2: Unsupervised Learning

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.

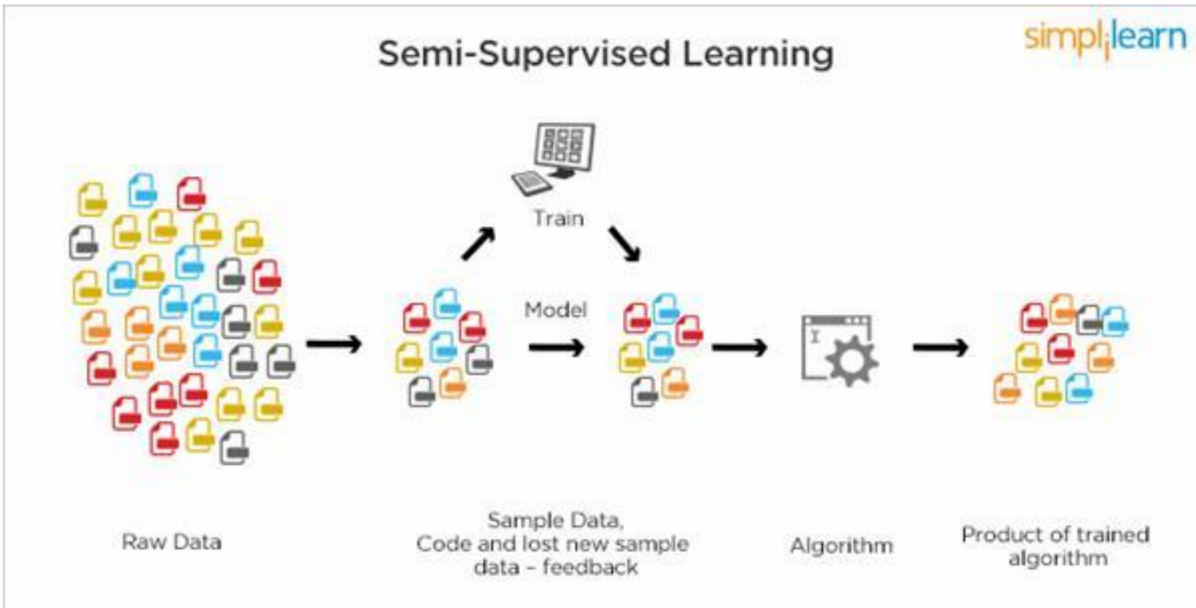


Figure 3: Semi Supervised Learning

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 PYHTON:

- 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.

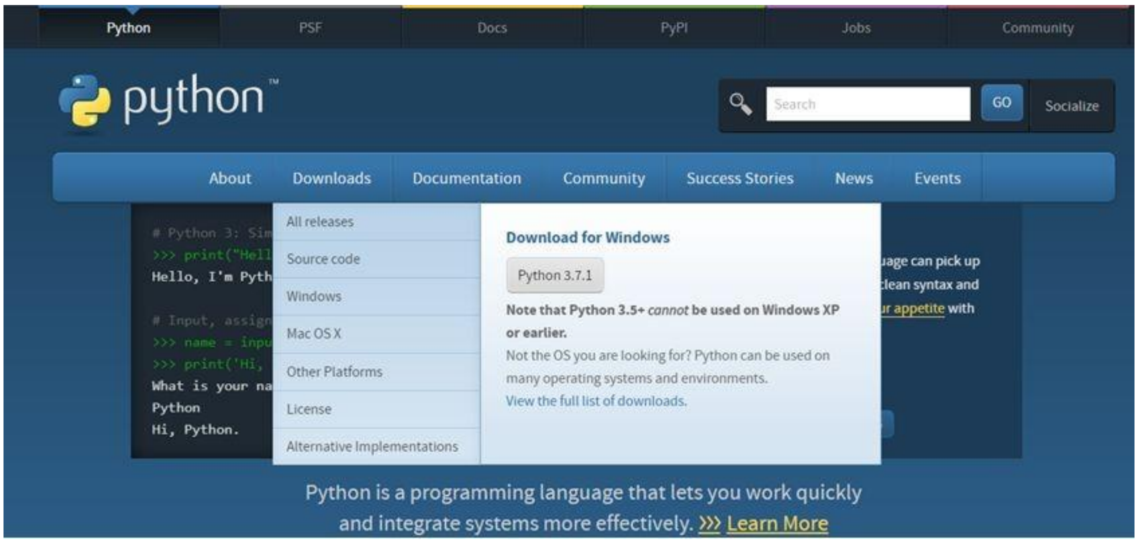


Figure 4: Python download

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)

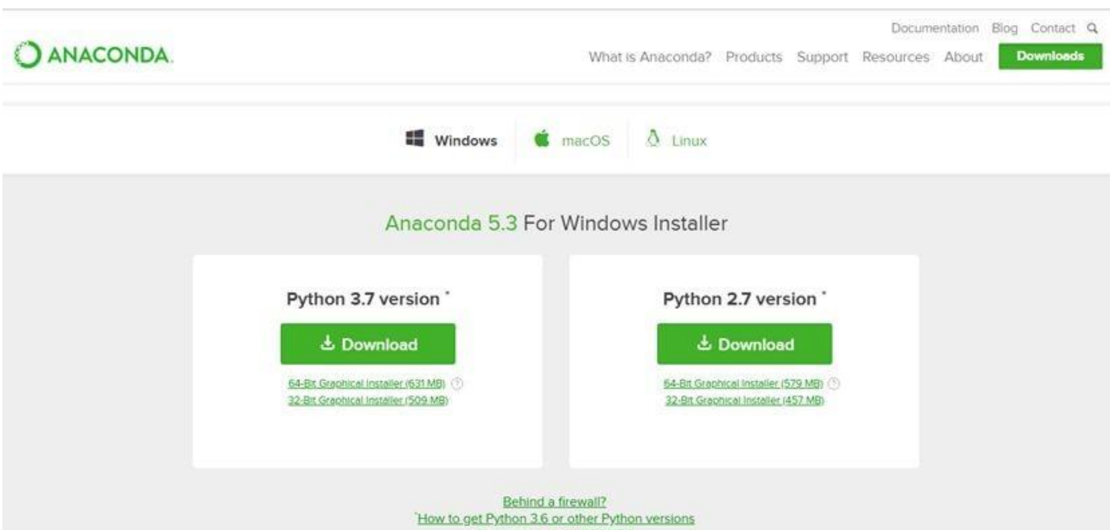


Figure 5: Anaconda download

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.

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

```python
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:

To predict the job posting is a real or fake posting using Machine Learning.

3.2 DATA SET:

The given data set consists of the following parameters:

1. Title
2. Description
3. Company profile
4. Requirements
5. Telecommuting
6. Has_company_logo
7. Has_questions
8. Employment_type
9. Required_experience
10. Required_education
11. Industry
12. Function
13. Fraudulent
14. City
15. Country_name: Name of the country mentioned in the job posting

3.3 OBJECTIVE OF THE CASE STUDY:

First, we will visualize the insights from the fake and real job advertisement and then we will use the Tfidf Vectorizer in this task and after successful training. Finally, we will evaluate the performance of our classifier using several evaluation metrics.

CHAPTER 4

MODEL BUILDING

4.1 PREPROCESSING OF THE DATA:

Preprocessing of the data actually involves the following steps:

4.1.1 GETTING THE DATASET:

- We can get the data set from the database or we

  can get the data from client.

4.1.2 IMPORTING THE LIBRARIES:

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

Importing Libraries

```
In [1]: import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
```

Figure 8 : Importing Libraries

4.1.3 IMPORTING THE DATA-SET:

Pandas in python provide an interesting method read_csv(). The read_csv function reads the entire dataset from a comma separated values file and we can assign it to a DataFrame to which all the operations can be performed. It helps us to access each and every row as well as columns and each and every value can be access using the dataframe. Any missing value or NaN value have to be cleaned.

- We are loading the data by using read_csv.

Data Loading

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

Figure 9 : Reading the dataset

Shape

The shape property returns a tuple representing the dimensionality of the Data Frame. The format of shape would be (rows, columns).

- We are looking at shape of dataset and top four columns.

```
In [3]: print(data.shape)
(17880, 18)

In [4]: data.head()
```

Out[4]:

	job_id	title	location	department	salary_range	company_profile	description	requirements	benefits	telecommu
0	1	Marketing Intern	US, NY, New York	Marketing	NaN	We're Food52, and we've created a groundbreaki...	Food52, a fast-growing, James Beard Award-winn...	Experience with content management systems a m...	NaN	
1	2	Customer Service - Cloud Video Production	NZ, , Auckland	Success	NaN	90 Seconds, the worlds Cloud Video Production ...	Organised - Focused - Vibrant - Awesome!Do you...	What we expect from you:Your key responsibilit...	What you will get from usThrough being part of...	
2	3	Commissioning Machinery Assistant (CMA)	US, IA, Wever	NaN	NaN	Valor Services provides Workforce Solutions th...	Our client, located in Houston, is actively se...	Implement pre-commissioning and commissioning ...	NaN	
3	4	Account Executive - Washington DC	US, DC, Washington	Sales	NaN	Our passion for improving quality of life thro...	THE COMPANY: ESRI – Environmental Systems Rese...	EDUCATION: Bachelor's or Master's in GIS, busi...	Our culture is anything but corporate—we have ...	
4	5	Bill Review Manager	US, FL, Fort Worth	NaN	NaN	SpotSource Solutions LLC is a Global Human Cap...	JOB TITLE: Itemization Review ManagerLOCATION:...	QUALIFICATIONS:RN license in the State of Texa...	Full Benefits Offered	

Fig-10 shape of data

4.1.4 Statistical Analysis:

Total Number or "N", Mean, Median, Mode and Standard Deviation are used to describe your data.

- The Total Number or "N" is the number of observations made.
- Mean: This is the average of the data. Adding the values of all of the observations and dividing the total by the total number of observations or "N".
- Median: This is the middle value of the observations.
- Mode: This is the most frequent observation**.**

```
In [5]: data.describe()
Out[5]:
```

	job_id	telecommuting	has_company_logo	has_questions	fraudulent
count	17880.000000	17880.000000	17880.000000	17880.000000	17880.000000
mean	8940.500000	0.042897	0.795302	0.491723	0.048434
std	5161.655742	0.202631	0.403492	0.499945	0.214688
min	1.000000	0.000000	0.000000	0.000000	0.000000
25%	4470.750000	0.000000	1.000000	0.000000	0.000000
50%	8940.500000	0.000000	1.000000	0.000000	0.000000
75%	13410.250000	0.000000	1.000000	1.000000	0.000000
max	17880.000000	1.000000	1.000000	1.000000	1.000000

```
In [6]: data.columns
Out[6]: Index(['job_id', 'title', 'location', 'department', 'salary_range',
         'company_profile', 'description', 'requirements', 'benefits',
         'telecommuting', 'has_company_logo', 'has_questions', 'employment_type',
         'required_experience', 'required_education', 'industry', 'function',
         'fraudulent'],
        dtype='object')
```

Fig 11:analysis of data

Unique:

The unique() function is used to find the unique elements of an array. Returns the sorted unique elements of an array.

Isnull:

isnull() function detect missing values in the given series object. It return a boolean same-sized object indicating if the values are NA.

We are checking for unique values and if any null values are present.If any null are found we use fillna() and imputation to fill columns with a data.

```
In [7]: data.nunique()

Out[7]: job_id                 17880
        title                  11231
        location                3105
        department              1337
        salary_range             874
        company_profile         1709
        description            14801
        requirements           11968
        benefits                6205
        telecommuting              2
        has_company_logo           2
        has_questions              2
        employment_type            5
        required_experience        7
        required_education        13
        industry                 131
        function                  37
        fraudulent                 2
        dtype: int64
```

```
In [8]: data.isnull().sum()

Out[8]: job_id                     0
        title                      0
        location                 346
        department             11547
        salary_range           15012
        company_profile         3308
        description                1
        requirements            2695
        benefits                7210
        telecommuting              0
        has_company_logo           0
        has_questions              0
        employment_type         3471
        required_experience     7050
        required_education      8105
        industry                4903
        function                6455
        fraudulent                 0
        dtype: int64
```

Fig 12: checking unique and null values

4.1.5 HANDLING MISSING VALUES:

Missing values can be handled in many ways using some inbuilt methods:

(a)dropna()

(b)fillna()

(c)interpolate()

(d)mean imputation and median imputation

(a)dropna():

- dropna() is a function which drops all the rows and columns which are having the missing values(i.e. NaN)

- dropna() function has a parameter called how which works as follows

- if how = ’all’ is passed then it drops the rows where all the columns of the particular row are missing

- if how = ’any’ is passed then it drops the rows where all the columns of the particular row are missing.

```
In [4]: data.head()
```

Out[4]:

	job_id	title	location	department	salary_range	company_profile	description	requirements	benefits	telecommuting	has
0	1	Marketing Intern	US, NY, New York	Marketing	NaN	We're Food52, and we've created a groundbreaki...	Food52, a fast-growing, James Beard Award-winn...	Experience with content management systems a m...	NaN	0	
1	2	Customer Service - Cloud Video Production	NZ, , Auckland	Success	NaN	90 Seconds, the worlds Cloud Video Production ...	Organised - Focused - Vibrant - Awesome!Do you...	What we expect from you:Your key responsibilit...	What you will get from usThrough being part of...	0	
2	3	Commissioning Machinery Assistant (CMA)	US, IA, Wever	NaN	NaN	Valor Services provides Workforce Solutions th...	Our client, located in Houston, is actively se...	Implement pre-commissioning and commissioning ...	NaN	0	
3	4	Account Executive - Washington DC	US, DC, Washington	Sales	NaN	Our passion for improving quality of life thro...	THE COMPANY: ESRI – Environmental Systems Rese...	EDUCATION: Bachelor’s or Master’s in GIS, busi...	Our culture is anything but corporate—we have ...	0	
4	5	Bill Review Manager	US, FL, Fort Worth	NaN	NaN	SpotSource Solutions LLC is a Global Human Cap...	JOB TITLE: Itemization Review ManagerLOCATION:...	QUALIFICATIONS:RN license in the State of Texa...	Full Benefits Offered	0	

Fig 13:before droping data

- We are dropping the unnecessary data which is not required.

```
In [9]: #droping the unnecessary columns
        data = data.drop(["job_id","salary_range"], axis=1)
```

```
In [10]: data.head()
```

Out[10]:

	title	location	department	company_profile	description	requirements	benefits	telecommuting	has_company_logo	has_
0	Marketing Intern	US, NY, New York	Marketing	We're Food52, and we've created a groundbreaki...	Food52, a fast-growing, James Beard Award-winn...	Experience with content management systems a m...	NaN	0	1	
1	Customer Service - Cloud Video Production	NZ, , Auckland	Success	90 Seconds, the worlds Cloud Video Production ...	Organised - Focused - Vibrant - Awesome!Do you...	What we expect from you:Your key responsibilit...	What you will get from usThrough being part of...	0	1	
2	Commissioning Machinery Assistant (CMA)	US, IA, Wever	NaN	Valor Services provides Workforce Solutions th...	Our client, located in Houston, is actively se...	Implement pre-commissioning and commissioning ...	NaN	0	1	
3	Account Executive - Washington DC	US, DC, Washington	Sales	Our passion for improving quality of life thro...	THE COMPANY: ESRI – Environmental Systems Rese...	EDUCATION: Bachelor's or Master's in GIS, busi...	Our culture is anything but corporate—we have ...	0	1	
4	Bill Review Manager	US, FL, Fort Worth	NaN	SpotSource Solutions LLC is a Global Human Cap...	JOB TITLE: Itemization Review ManagerLOCATION:...	QUALIFICATIONS:RN license in the State of Texa...	Full Benefits Offered	0	1	

Figure 14: data after using drop()

(b)fillna():

- fillna() is a function which replaces all the missing values using different ways.
- if we use method = ’ffill’ where ffill is a method called forward fill, which carry forwards the previous row’s value
- if we use method = ‘bfill’ where bfill is a method called backward fill, which carry backward the next row’s value
- if we use method = ‘ffill’ , axis = ‘columns’ then it carry forwards the previous column’s value

- if we use method = ‘bfill’ , axis = ‘columns’ then it carry backward the next column’s value.

(c)interpolate():

- interpolate() is a function which comes up with a guess value based on the other values in the dataset and fills those guess values in the place of missing values

(d)mean and median and mode imputation

- mean and median and mode imputation can be performed by using fillna().

- mean imputation calculates the mean for the entire column and replaces the missing values in that column with the calculated mean.

- median imputation calculates the median for the entire column and replaces the missing values in that column with the calculated median
- mode imputation calculates the median for the entire column and replaces the missing values in that column with the calculated mode
- We are filling Nan values with mode imputation.

```
In [11]: data['title'].fillna(data['title'].mode()[0],inplace=True)

In [12]: data['location'].fillna(data['location'].mode()[0],inplace=True)

In [13]: data['department'].fillna(data['department'].mode()[0],inplace=True)

In [14]: data['company_profile'].fillna(data['company_profile'].mode()[0],inplace=True)

In [15]: data['description'].fillna(data['description'].mode()[0],inplace=True)

In [16]: data['requirements'].fillna(data['requirements'].mode()[0],inplace=True)

In [17]: data['benefits'].fillna(data['benefits'].mode()[0],inplace=True)

In [18]: data['telecommuting'].fillna(data['telecommuting'].mode()[0],inplace=True)

In [19]: data['has_company_logo'].fillna(data['has_company_logo'].mode()[0],inplace=True)

In [20]: data['has_questions'].fillna(data['has_questions'].mode()[0],inplace=True)

In [21]: data['employment_type'].fillna(data['employment_type'].mode()[0],inplace=True)

In [22]: data['required_experience'].fillna(data['required_experience'].mode()[0],inplace=True)
```

```
In [23]: data['required_education'].fillna(data['required_education'].mode()[0],inplace=True)

In [24]: data['industry'].fillna(data['industry'].mode()[0],inplace=True)

In [25]: data['function'].fillna(data['function'].mode()[0],inplace=True)

In [26]: data['fraudulent'].fillna(data['fraudulent'].mode()[0],inplace=True)

In [27]: data.head()
```

Figure : Mode Imputation

5.Generating Plots

5.1 -Visualize the data between Target and the Features

Count plot

A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable. The basic API and options are identical to those for **barplot()**, so you can compare counts across nested variables.

Input data can be passed in a variety of formats, including:

- Vectors of data represented as lists, numpy arrays, or pandas Series objects passed directly to the x, y, and/or hue parameters.
- A “long-form” DataFrame, in which case the x, y, and hue variables will determine how the data are plotted.
- A “wide-form” DataFrame, such that each numeric column will be plotted.
- An array or list of vectors.

Exploratory Data Analysis

Data visualization

```
import seaborn as sns
sns.countplot(data.fraudulent).set_title('Real & Fradulent')
data['fraudulent'].value_counts()
```

```
0    17014
1      866
Name: fraudulent, dtype: int64
```

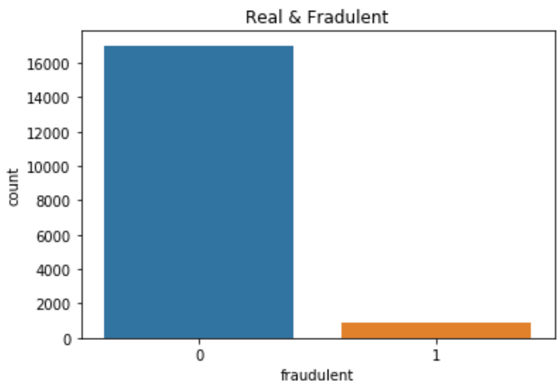


Fig 15 Data visualize

- we will check the total number of fraudulent postings and real postings.

5.2 - Visualize the data between all the Features

Bar plot:

A bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally.

A bar graph shows comparisons among discrete categories. One axis of the chart shows the specific categories being compared, and the other axis represents a measured value.

Matplotlib API provides the **bar()** function that can be used in the MATLAB style use as well as object oriented API.

- We are using some plots to visualize the data and perform operations

```
: required_education  = dict(data.required_education.value_counts()[:10])
  plt.figure(figsize=(25,15))
  plt.title('frequency of required_education', size=20)
  plt.bar(required_education.keys(), required_education.values())
  plt.ylabel('frequency', size=20)
  plt.xlabel('required_education', size=20)
```

```
: Text(0.5, 0, 'required_education')
```

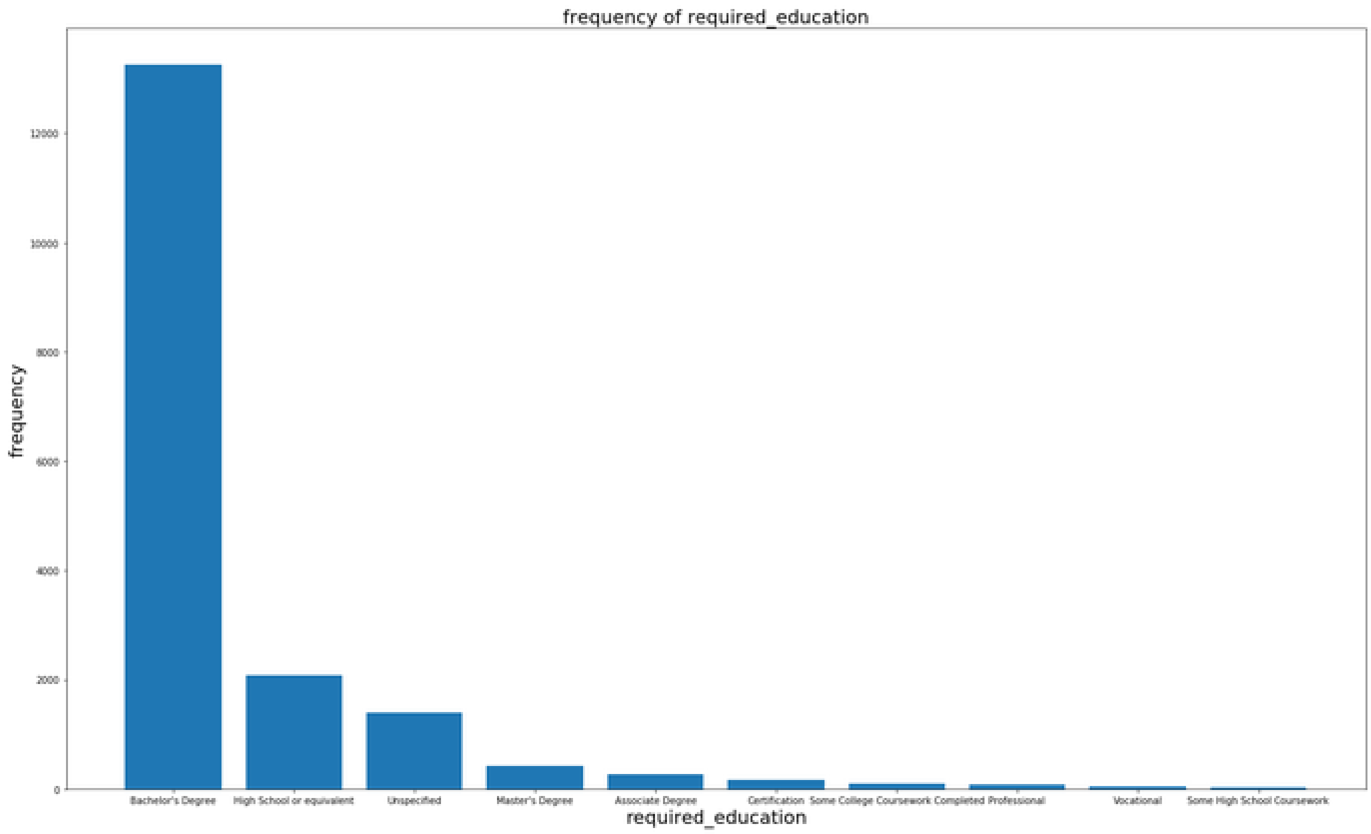


Fig 16 Frequency vs required education

In this plot, we will visualize the number of job postings by required education. And we can observe Bachelor Degree have high frequency compared to other types.

```
: employment_type  = dict(data.employment_type.value_counts()[:11])
plt.figure(figsize=(25,15))
plt.title('frequency of employment_type', size=20)
plt.bar(employment_type.keys(), employment_type.values())
plt.ylabel('frequency', size=20)
plt.xlabel('employment_type', size=20)
```

```
: Text(0.5, 0, 'employment_type')
```

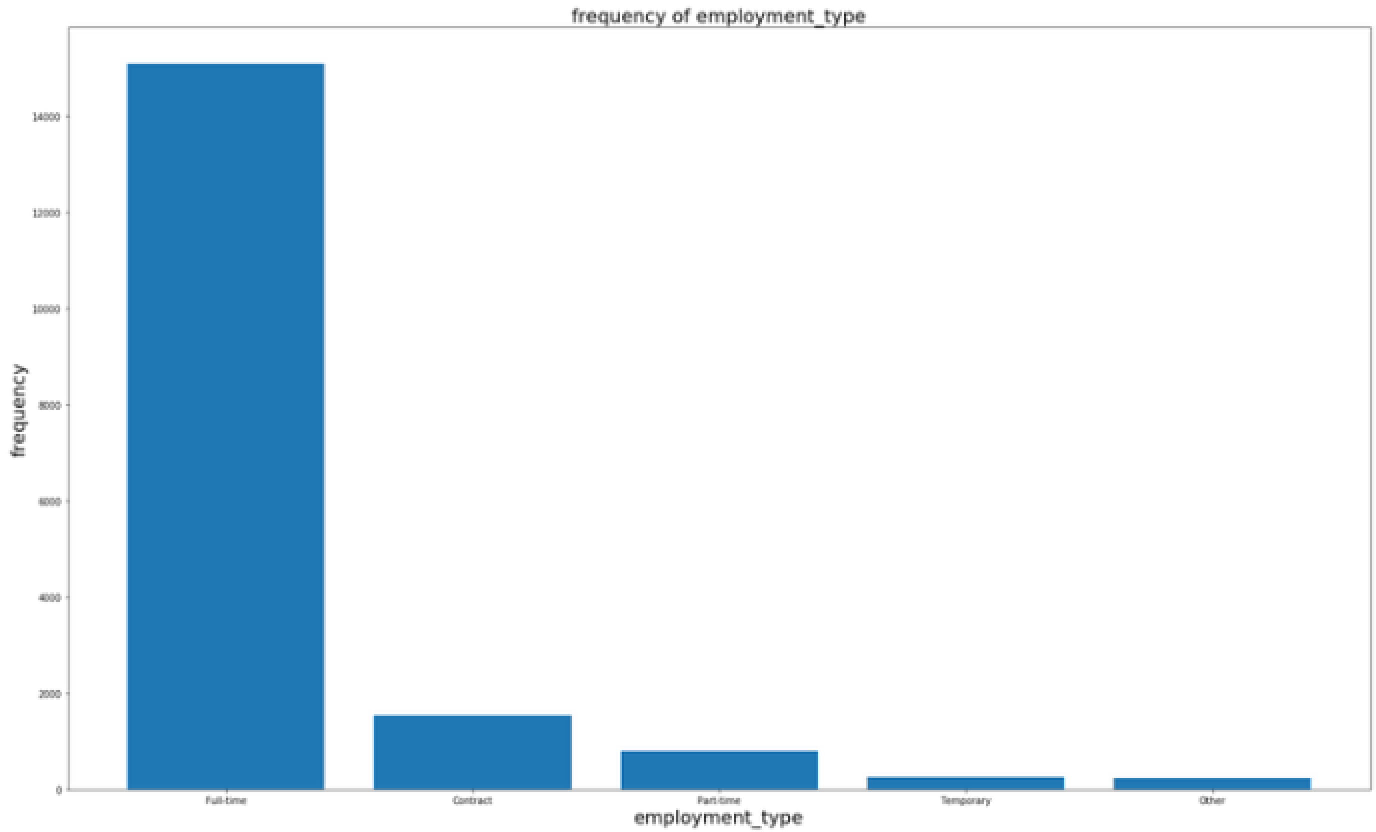


Fig 17 Frequency vs Employment type

In this plot, we will visualize the number of job postings by employment type. And we can observe Full time have high frequency compared to other types.

```python
required_experience  = dict(data.required_experience.value_counts()[:11])
plt.figure(figsize=(25,15))
plt.title('frequency of required_experience', size=20)
plt.bar(required_experience.keys(), required_experience .values())
plt.ylabel('frequency', size=20)
plt.xlabel('required_experience', size=20)
```

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

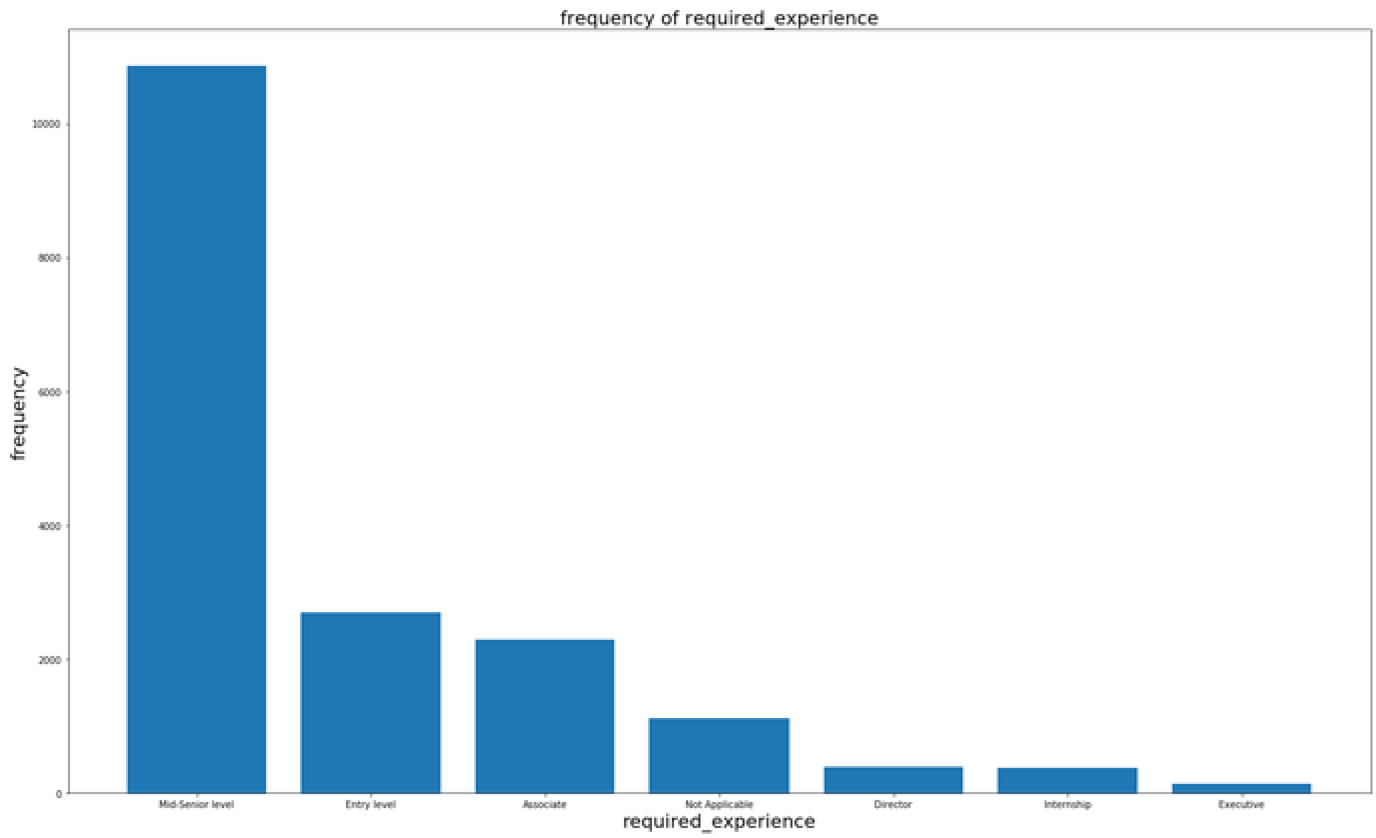


Fig 18 Frequency vs required experience

In this plot, we will visualize the number of job postings by required experience. And we can observe Mid-Senior level have high frequency compared to other types.

```
function = dict(data.function.value_counts()[:11])
plt.figure(figsize=(25,15))
plt.title('frequency of function', size=20)
plt.bar(function.keys(), function .values())
plt.ylabel('frequency', size=20)
plt.xlabel('function', size=20)
```

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

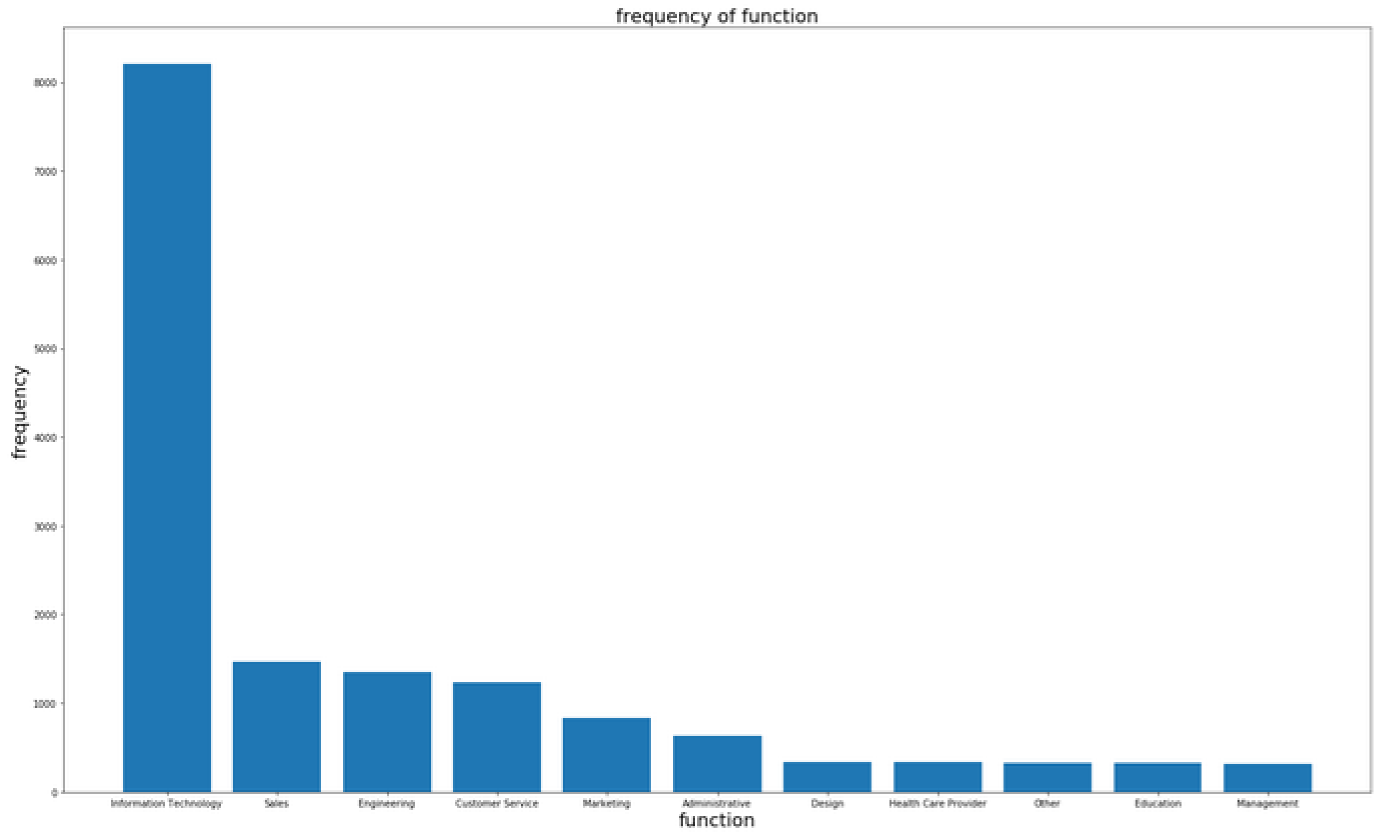


Fig 19 Frequency vs function

In this plot, we will visualize the number of job postings by function. And we can observe Information Technology have high frequency compared to other types.

```
industry  = dict(data.industry.value_counts()[:11])
plt.figure(figsize=(25,15))
plt.title('frequency of industry', size=20)
plt.bar(function.keys(), function.values())
plt.ylabel('frequency', size=20)
plt.xlabel('industry', size=20)
```

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

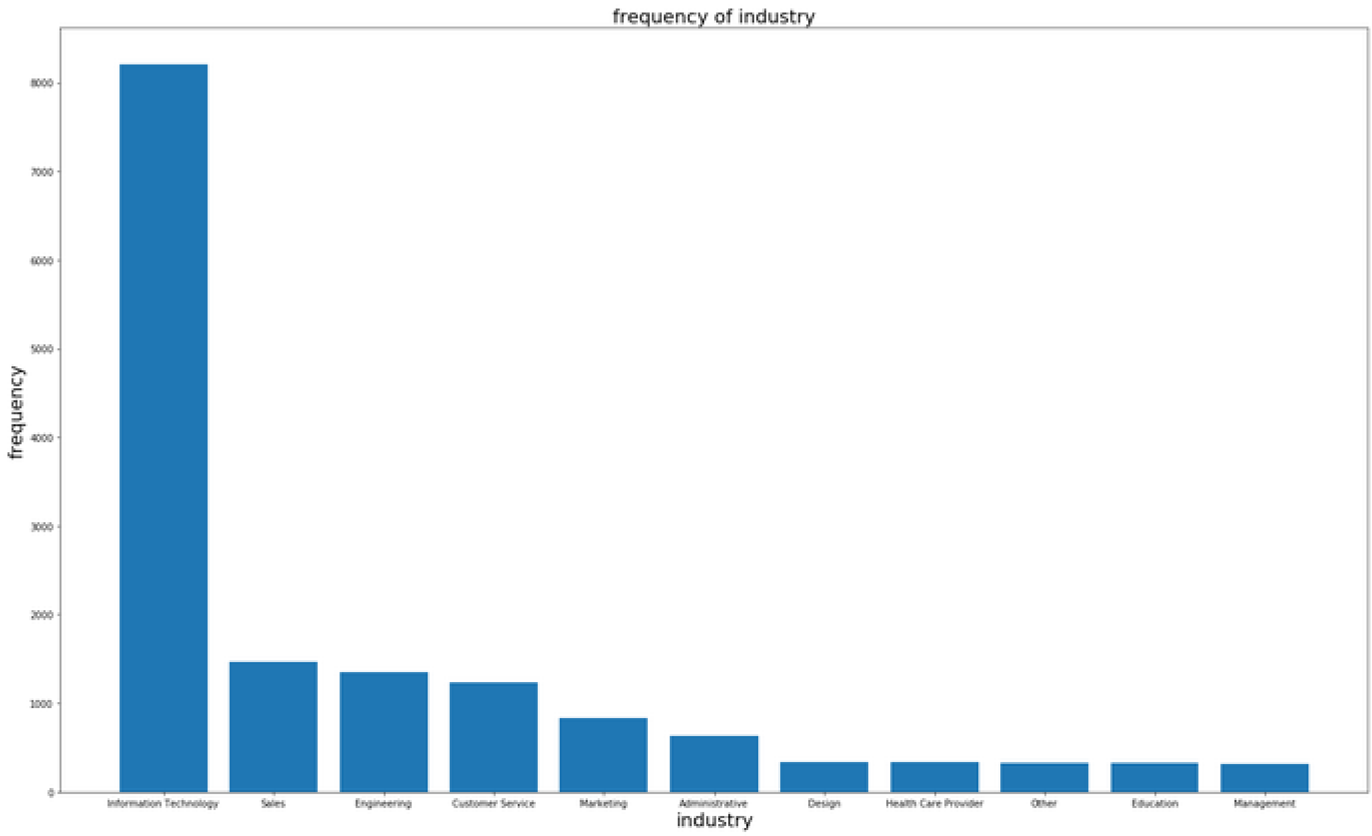


Fig 20 Frequency vs required industry

In this plot, we will visualize the number of job postings by Industry. And we can observe Information Technology have high frequency compared to other types.

```
sns.countplot(data.has_questions).set_title('If there was an recruitment process or not')
data['has_questions'].value_counts()
```

```
0    9088
1    8792
Name: has_questions, dtype: int64
```

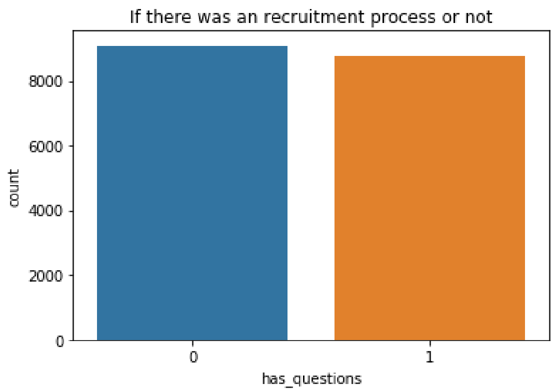


Fig 21 have any recruitment process or not

In this plot, we will visualize the number of job postings have any recruitment process or not. And we can observe have high frequency for having.

```
sns.countplot(data.telecommuting).set_title('If there was an communication process or not')
data['telecommuting'].value_counts()
```

```
0    17113
1      767
Name: telecommuting, dtype: int64
```

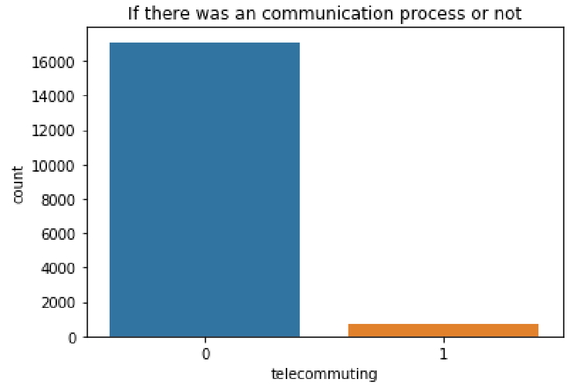


Fig 22 have any communication process or not

In this plot, we will visualize the number of job postings have any telecommuting or not. And we can observe have high frequency for having.

```
sns.countplot(data.has_company_logo).set_title('True if the company logo is present')
data['has_company_logo'].value_counts()
```

```
1    14220
0     3660
Name: has_company_logo, dtype: int64
```

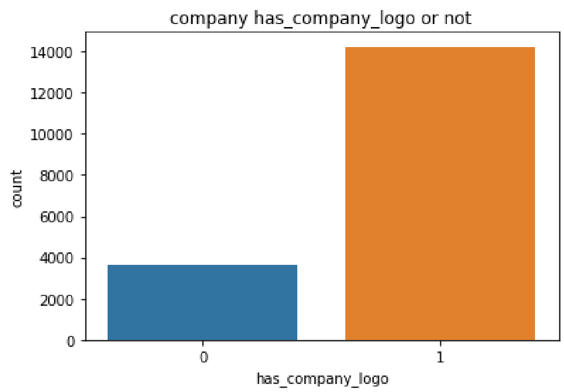


Fig 23 have any company logo or not

In this plot, we will visualize the number of job postings have any have a company logo or not. And we can observe have less frequency for having.

Heatmap:

A **heat map** (or **heatmap**) is a data visualization technique that shows magnitude of a phenomenon as color in two dimensions. The variation in color may be by hue or intensity, giving obvious visual cues to the reader about how the phenomenon is clustered or varies over space.

```
corelation = data.corr()
```

```
sns.heatmap(corelation, xticklabels=corelation.columns, yticklabels=corelation.columns, annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xcf1afc8>
```

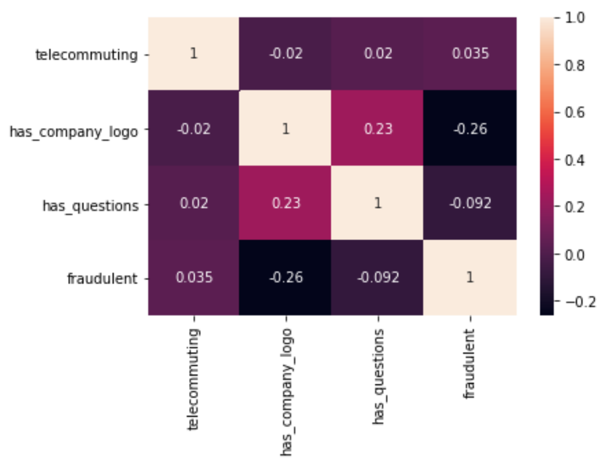


Fig 24 heatmap()

6.TRAINING THE MODEL:

- 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)

- We are Spliting arrays or matrices into random train and test subsets .

```
data['function'] = data['title'] + ' ' + data['location'] + ' ' + data['department'] + ' ' + data['company_profile'] + ' ' + c
```

```
X = data.function
y=data.fraudulent
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=2)
```

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

```
(12516,)
(5364,)
(12516,)
(5364,)
```

Fig 25 Train_Test_Split

7. TFIDF Vectorizer

Word counts are a good starting point, but are very basic.
One issue with simple counts is that some words like "*the*" will appear many times and their large counts will not be very meaningful in the encoded vectors.
An alternative is to calculate word frequencies, and by far the most popular method is called TF-IDF. This is an acronym that stands for "*Term Frequency – Inverse Document*" Frequency which are the components of the resulting scores assigned to each word.

- **TF:** Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more often in long documents than shorter ones**.**

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

- **Inverse Document Frequency**: This downscales words that appear a lot across documents**,** which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus, we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

IDF(t) = log(Total number of documents / Number of documents with term t in it).

The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents. Alternatively, if you already have a learned CountVectorizer, you can use it with a TfidfTransformer to just calculate the inverse document frequencies and start encoding documents.

The **term frequency**, the number of times a term occurs in a given document, is multiplied with **idf** component, which is computed as **idf** is the inverse *document* frequency, so it's the ratio of the number of *documents* (all documents vs documents that contain the term at least once).

Example:
Consider a document containing 100 words wherein the word cat appears 3 times.
The term frequency (i.e., TF) for cat is then (3 / 100) = 0.03. Now, assume we have 10 million documents and the word cat appears in one thousand of these. Then, the inverse document frequency (i.e., IDF) is calculated as log(10,000,000 / 1,000) = 4.Thus, the Tf-idf weight is the product of these quantities: 0.03 * 4 = 0.12.

- We are using Tfidf vectorizer which will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents

```
# TFIDF Vectorizer
# Import TfIDF Vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize and object for the TFIDF Vectorizer
tfidf = TfidfVectorizer()
```

```
# apply the tfidf to the data(x_train)
X_train_transformed = tfidf.fit_transform(X_train)
X_train_transformed
```

```
<12516x86299 sparse matrix of type '<class 'numpy.float64'>'
	with 2776752 stored elements in Compressed Sparse Row format>
```

```
X_test_transformed = tfidf.transform(X_test)
X_test_transformed
```

```
<5364x86299 sparse matrix of type '<class 'numpy.float64'>'
	with 1171516 stored elements in Compressed Sparse Row format>
```

Fig 26 Tdif Vectorizier

```python
# Feature Names
tfidf.get_feature_names()
```

```
['00',
 '000',
 '0000',
 '0001pt',
 '0005',
 '000a',
 '000aed',
 '000applying',
 '000benefits',
 '000bonus',
 '000cash',
 '000commission',
 '000company',
 '000equity',
 '000full',
 '000gbp',
 '000generate',
 '000health',
 '000highly',
```

Fig 27 Feature Name

```python
# position of the words in the sparse matrix
tfidf.vocabulary_
```

```
{'caregiver': 11634,
 'hha': 33602,
 'cna': 13788,
 'watervliet': 81497,
 'hartford': 33109,
 'us': 79696,
 'mi': 45041,
 'sales': 64515,
 'our': 50575,
 'mission': 45456,
 'to': 75720,
 'clients': 13369,
 'is': 37917,
 'preserve': 55964,
 'their': 74901,
 'independence': 35709,
 'enhance': 25124,
 'quality': 59175,
 'of': 48643,
```

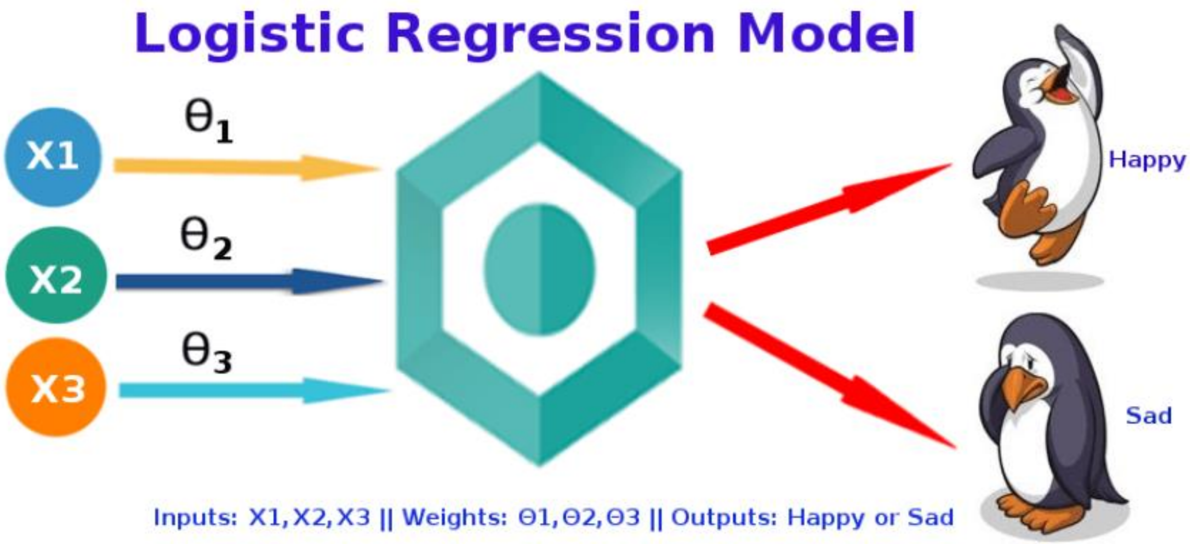
```python
tfidf.idf_
```

```
array([4.67594123, 3.45105303, 9.74169582, ..., 9.74169582, 9.04854864,
       9.74169582])
```

Fig 28 tfidf vocabulary

8. MODEL BUILDING AND EVALUATION:

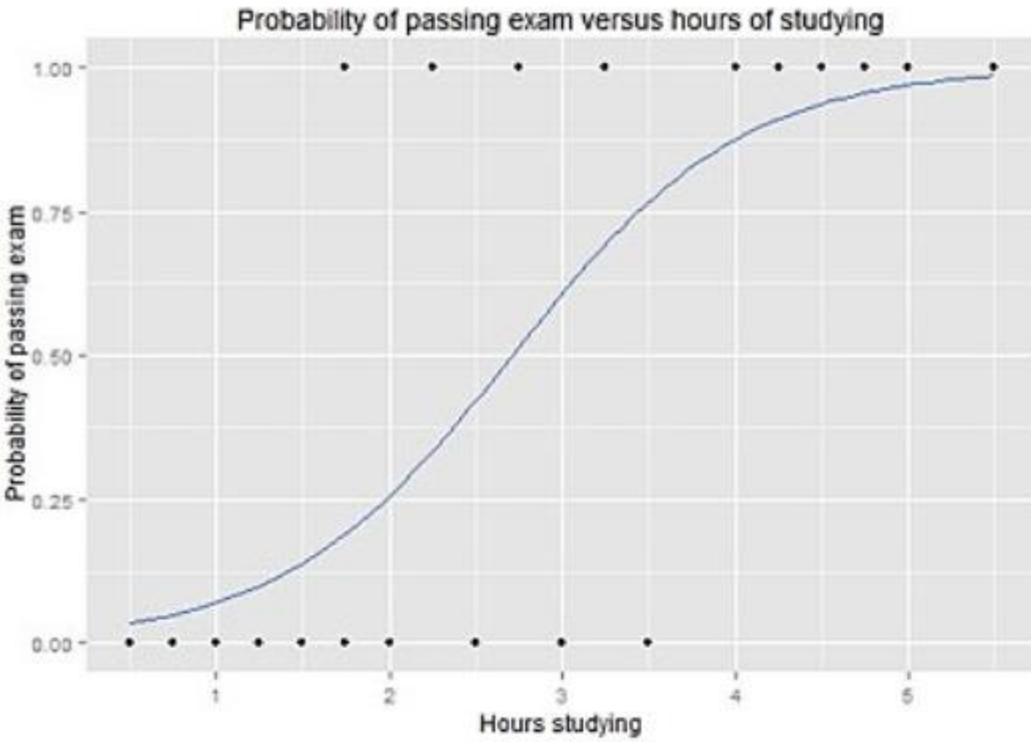
8.1 Logistic Regression



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.

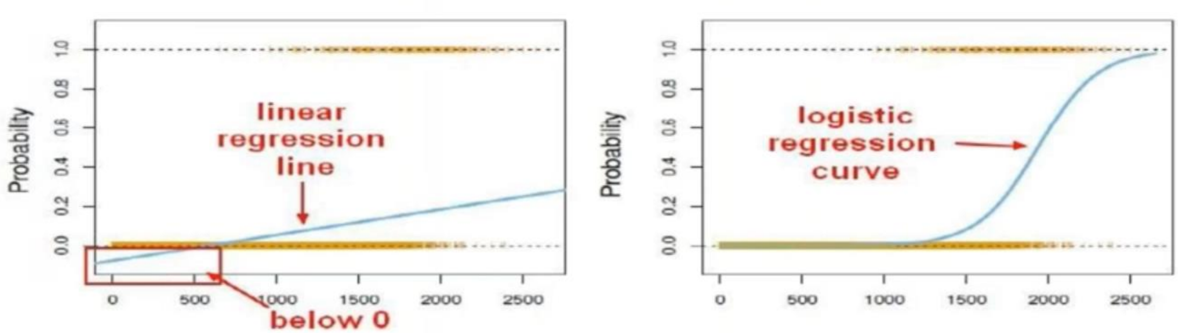


Logistic Regression

- Logistic Regression model predicts the probability associated with each dependent variable Category.

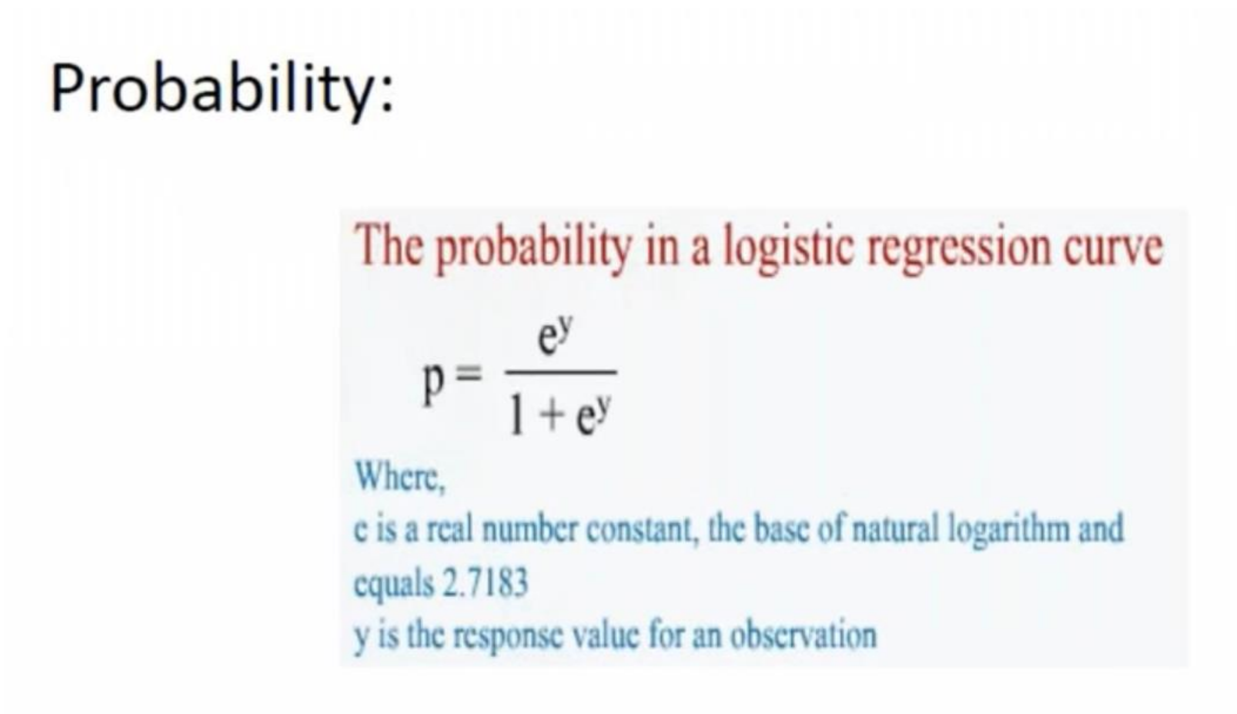
How does it do this?

- It finds linear relationship between independent variables and a link function of this probabilities. Then the link function that provides the best goodness-of-fit for the given data is chosen



$$Z = b0 + b1(x1) + b2(x2) + b3(x3)$$

But, when we use the above equation to calculate probability, we would get values less than 0 as well as greater than 1. That doesn't make any sense. So, we need to use such an equation which always gives values between 0 and 1, as we desire while calculating the probability.
Out of the equation we are going to calculate the probabilities of the categories.



The final step is to assign class labels (0 or 1) to our predicted probabilities.
If p is less than 0.5, we conclude the predicted output is 0 and if p is greater than 0.5, you conclude the output is 1.
Methods:

There are three methods of Logistic Regression based on nature of the attribute data.

- Binary
- Nominal
- Ordinal

✓ Binary Logistic Regression

✓ Binary logistic Regression is performed on the Binary response variables. It has only two categories, such as presence or absence of disease, pass or fail, defective or non-defective products.

✓ Nominal Logistic Regression

✓ Nominal Logistic Regression is performed on the Nominal variables. These are categorical variables that have three or more possible categories with no natural ordering

Example: Food is crunchy, mushy and crispy

✓ Ordinal Logistic Regression

✓ Ordinal Logistic Regression is performed on ordinal response variables. These are categorical variable that have three or more possible categories with a natural ordering.

Example: Survey on quality of a shirt material; strongly disagree, disagree, neutral, agree and strongly agree.

Method	Description of categorical response variable	Example
Binary	Two categories	Presence/absence of disease
Nominal	Three or more categories with no natural ordering to the levels	Crunchy/mushy/ crispy
Ordinal	Three or more categories with ordering of the levels	Strongly disagree/ disagree/neutral/ agree/strongly agree

8.1.1 Train and Test Models:

- Importing LogisticRegression from packages of linear_model.

  The training dataset is used to prepare a model, to train it.

- We pretend the test dataset is new data where the output values are withheld from the algorithm. We gather predictions from the trained model on the inputs from the test dataset and compare them to the withheld output values of the test set.

```
## Import Logistic Regression
from sklearn.linear_model import LogisticRegression
# creating an object for Logistic Regression
Lr=LogisticRegression()
```

```
Lr.fit(X_train_transformed, y_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)
```

```
Lr.fit(X_test_transformed, 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)
```

Fig 29: Train the model

8.1.2 Predict Analytics

Predictive analytics uses historical data to predict future events. Typically, historical data is used to build a mathematical model that captures important trends. That predictive model is then used on current data to predict what will happen next, or to suggest actions to take for optimal outcomes.

```
y_train_pred = Lr.predict(X_train_transformed)
y_train_pred
```

...

```
y_test_pred = Lr.predict(X_test_transformed)
y_test_pred
```

Fig 30: Predict Values

8.1.3 Classification report

A Classification report is used to measure the quality of predictions from a classification algorithm. How many predictions are True and how many are False. More specifically, True Positives, False Positives, True negatives and False Negatives are used to predict the metrics of a classification report as shown below.

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

```
array([[11903,     2],
       [  348,   263]], dtype=int64)
```

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

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.99     11905
           1       0.99      0.43      0.60       611

    accuracy                           0.97     12516
   macro avg       0.98      0.72      0.79     12516
weighted avg       0.97      0.97      0.97     12516
```

Fig 31: Classification Report

```
# 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)
```

```
array([[5109,    0],
       [ 177,   78]], dtype=int64)
```

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

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.98      5109
           1       1.00      0.31      0.47       255

    accuracy                           0.97      5364
   macro avg       0.98      0.65      0.73      5364
weighted avg       0.97      0.97      0.96      5364
```

Fig 32: Classification Report

8.1.4 Accuracy score

Accuracy is one metric for evaluating classification models. Informally, **accuracy** is the fraction of predictions our model got right.

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

```
0.9720357941834452
```

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

```
0.964765100671141
```

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

```
96.47651006711409
```

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

```
1.0
```

Fig 33: Accuracy Score

8.2 Naive Bayes

Naive Bayes Algorithm comes under Supervised Learning. It is a classification algorithm, which performs well on numerical and the text data. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. It is one of the simplest supervised learning algorithms. Naive Bayes classifier is the fast, accurate and reliable algorithm. Naive Bayes classifiers have high accuracy and speed on large datasets.
Naive Bayes Classifier assumes that the effect of a particular feature in a class is independent of other features. For example, a loan applicant is desirable or not depending on his/her income, previous loan and transaction history, age, and location. Even if these features are interdependent, these features are still considered independently. This assumption simplifies computation, and that's why it is considered as 'Naive'. This assumption is called class conditional independence.

How to build a basic model using Naive Bayes in Python

Again, scikit learn (python library) will help here to build a Naive Bayes model in Python. There are three types of Naive Bayes model under the scikit-learn library:

- **Gaussian:** It is used in classification and it assumes that features follow a normal distribution. Because of the assumption of the normal distribution, Gaussian Naive Bayes is used in cases when all our features are continuous.

- **Bernoulli:** The binomial model is useful if your feature vectors are binary (i.e. zeros and ones). One application would be text classification with a 'bag of words' model where the 1s & 0s are "word occurs in the document" and "word does not occur in the document" respectively.

- **Multinomial:** It is used for discrete counts. For example, let's say, we have a text classification problem. Here we can consider Bernoulli trials which is one step further and instead of "word occurring in the document", we have to "count how often word occurs in the document", you can think of it as "number of times outcome number x_i is observed over the n trials".

One of the major advantages that Naive Bayes has over other classification algorithms is its ability to handle an extremely large number of features. In our case, each word is treated as a feature and there are thousands of different words. Also, it performs well even with the presence of irrelevant features and is relatively unaffected by them. It rarely ever overfits the data. Another important advantage is that its model training and prediction times are very fast for the amount of data it can handle.

8.2.1 Train the Models

- Import BernoulliNB method which is available in package naive bayes from scikit learn library

- Once the model is built, we need to check for accuracy.

- This can be done using predict method which is used to predict the output for input test set, and compare the predicted output with original output test set.

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

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

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

Fig 34 Train

8.2.2 Predict Analytics

```
y_train_pred = model_BernNB.predict(X_train_transformed)
```

```
y_test_pred = model_BernNB.predict(X_test_transformed)
```

Fig 35 Predict Values

8.2.3 Classification Report

A Classification report is used to measure the quality of predictions from a classification algorithm. How many predictions are True and how many are False. More specifically, True Positives, False Positives, True negatives and False Negatives are used to predict the metrics of a classification report as shown below.

```
# 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([[11846,    59],
       [  444,   167]], dtype=int64)
```

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

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98     11905
           1       0.74      0.27      0.40       611

    accuracy                           0.96     12516
   macro avg       0.85      0.63      0.69     12516
weighted avg       0.95      0.96      0.95     12516
```

Fig 36 Classification Report

```
# 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)
```

```
array([[5108,    1],
       [ 252,    3]], dtype=int64)
```

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

```
              precision    recall  f1-score   support

           0       0.95      1.00      0.98      5109
           1       0.75      0.01      0.02       255

    accuracy                           0.95      5364
   macro avg       0.85      0.51      0.50      5364
weighted avg       0.94      0.95      0.93      5364
```

Fig 37 Classification Report

8.2.4 Accuracy Score

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

```
0.9598114413550655
```

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

```
0.9528337061894109
```

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

```
95.2647278150634
```

Fig 38Accuracy Score

9 Hyper Parameter 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

10 GridSearchCV

What is grid search?

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.

```
from sklearn.model_selection import GridSearchCV
dual=[False]
max_iter=[100]
param_grid = dict(dual=dual,max_iter=max_iter)
```

```
#Import the GridSearchCV
from sklearn.model_selection import GridSearchCV

# initialization of GridSearch with the parameters- ModelName and the dictionary of parameters
Lr = LogisticRegression(dual=False)
grid_search = GridSearchCV(estimator=Lr, param_grid=param_grid, cv = 3, n_jobs=-1)

# applying gridsearch onto dataset
grid_search.fit(X_train_transformed, y_train)
grid_result = grid_search.fit(X_train_transformed, y_train)
```

```
grid_result.best_params_
```

```
{'dual': False, 'max_iter': 100}
```

Fig 39: Hyper parameter for Logistic Regression

```
Lr = LogisticRegression(dual = False, max_iter = 100)

# We need to fit the model to the data
Lr.fit(X_train_transformed, y_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)
```

```
# Prediction on test data
pred_test = Lr.predict(X_test_transformed)

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

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.99      5109
           1       0.99      0.40      0.57       255

    accuracy                           0.97      5364
   macro avg       0.98      0.70      0.78      5364
weighted avg       0.97      0.97      0.97      5364
```

```
Lr_score = (Lr.score(X_test_transformed, pred_test))*100
Lr_score
```

```
100.0
```

Fig 40: classification Report

```
Methods = ['LogisticRegression', 'NaiveBayes']
Scores = np.array([Lr_score,naive_score])

fig, ax = plt.subplots(figsize=(8,6))
sns.barplot(Methods, Scores)
plt.title('Algorithm Prediction Accuracies')
plt.ylabel('Accuracy')
```

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

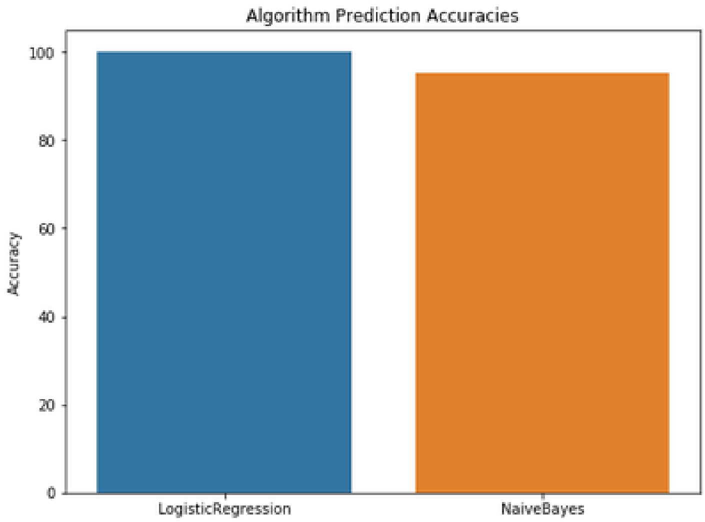


Fig 41:Comparsion plot after Hyper parameters Tuning GridsearchCV

Conclusion

```
Methods = ['LogisticRegression', 'NaiveBayes']
Scores = np.array([Lr_score,naive_score])

fig, ax = plt.subplots(figsize=(8,6))
sns.barplot(Methods, Scores)
plt.title('Algorithm Prediction Accuracies')
plt.ylabel('Accuracy')
```

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

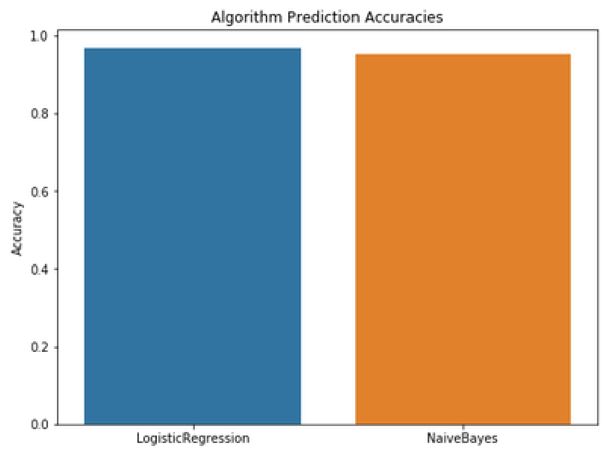


Fig 42: comparison plot(before gridsearchcsv)

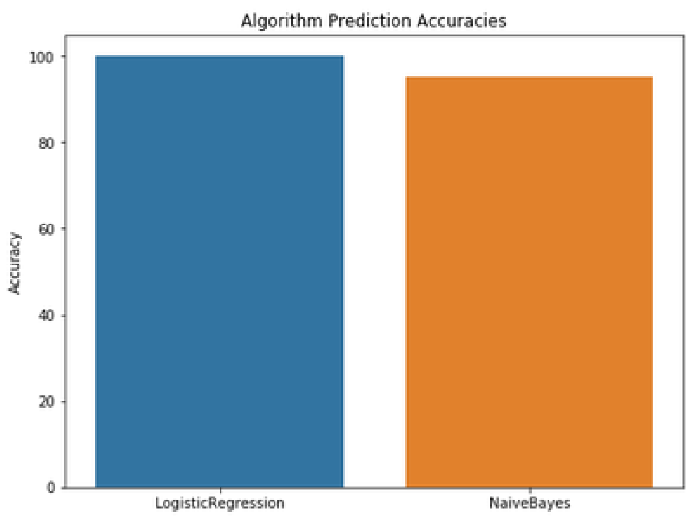


Fig 43:Comparison plot after Hyper parameters Tuning GridsearchCV

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) ,after seeing accuracy and comparison plot before and after gridsearchcv of Logistic Regression and Naïve Bayes and its come to point that using of Logistic regression is better rather than Naïve Bayes.

Reference

https://analyticsindiamag.com/classifying-fake-and-real-job-advertisements-using-machine-learning/

https://towardsdatascience.com/predicting-fake-job-postings-part-2-predictive-analysis-3119ba570c35