

PROFESSIONAL TRAINING REPORT
At
SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Submitted in partial fulfillment of the requirements for the
award of Bachelor of Engineering Degree in

Information Technology

By

Porkodi.S

(Reg. No. 39120080)



DEPARTMENT OF INFORMATION TECHNOLOGY
SCHOOL OF COMPUTING

**SATHYABAMA INSTITUTE OF
SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

(Established under Section 3 of UGC Act, 1956)

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600119

www.sathyabamauniversity.ac.in



NOVEMBER 2021



SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade “A” by NAAC
(Established under Section 3 of UGC Act, 1956)
JEPPIAAR NAGAR, RAJIV GANDHI SALAI
CHENNAI– 600119
www.sathyabama.ac.in



DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **PORKODI.S**
(Reg. No: 39120080) who carried out the project entitled “**VACCINE USAGE
PREDICTION USING MACHINE LEARNING**” under my supervision from
September 2021 to November 2021.

Internal Guide

Dr. L. Mary Gladence, M.E., Ph.D.,

Head of the Department

Dr. R. SUBHASHINI, M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, PORKODI.S (Reg. No. 39120120) hereby declare that the Project report entitled "**Vaccine Usage Prediction Using Machine Learning**" done by me under the guidance **Dr. L. Mary Gladence, M.E., Ph.D.**, is submitted in partial fulfillment of the requirements for the award of Bachelor of degree in Information Technology.

DATE:

PLACE:




SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. SASIKALA, M.E., Ph.D., Dean, School of Computing** and **Dr. SUBHASHINI, M.E., Ph.D., Head of the Department, Dept. of Information Technology** for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr .L. Mary Gladence, M.E., Ph.D.,** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **DEPARTMENT OF INFORMATION TECHNOLOGY** who were helpful in many ways for the completion of the project.

TRAINING CERTIFICATE



IMARTICUS
LEARNING

Certificate of Completion

Awarded to

PORKODLS

Upon successfully Completed the Boot Camp Training in Machine Learning
for 45 Hrs from 01st Sep 2021 To 27th Sep 2021





Nikhil Barshikar
Director

Imarticus Learning Private Limited

www.imarticus.org

ABSTRACT

Subjects receiving the same vaccine often show different levels of immune responses and some may even present adverse side effects to the vaccine. Systems vaccinology can combine omics data and machine learning techniques to obtain highly predictive signatures of vaccine immunogenicity and reactogenicity. Currently, several machine learning methods are already available to researchers with no background in bioinformatics. We propose a data-driven machine learning model to predict the likelihood of a person being vaccinated against H1N1 and seasonal flu. In 2009, a pandemic that was caused by the H1N1 influenza virus, named "swine flu", spread like wildfire across the world. Researchers and Healthcare specialists estimated that in the initial year, it was responsible for between 150,000 to 600,000 deaths in the whole world. A vaccine against the H1N1 (swine flu) virus was made available in October 2009. I have mainly used Logistic Regression Algorithm.

TABLE OF CONTENTS

Chapter No.	TITLE	Page No.
	ABSTRACT	i
	LIST OF FIGURES	ii
	LIST OF TABLES	iii
	LIST OF ABBREVIATIONS	iv
1.	INTRODUCTION	
	1.1 WHAT IS MACHINE LEARNING	1
	1.2 WHY IS MACHINE LEARNING IMPORTANT?	1
	1.3 MACHINE LEARNING HISTORY	1
	1.4 WHAT ARE THE DIFFERENCE TYPES OF MACHINE LEARNING	2
	1.5 SUPERVISED MACHINE LEARNING?	3
	1.6 LOGISTIC REGRESSION	4
2.	AIM AND SCOPE OF THE PRESENT INVESTIGATION	6
	2.1 AIM	6
	2.2 SCOPE	6
3.	EXPERIMENTAL OR MATERIALS AND METHODS, ALGORITHMS USED	7
	3.1 OVERVIEW	7
	3.2 REQUIREMENTS	7
	3.3 DATASET INFORMATION	7

	3.4 METHODOLOGY	10
4.	RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS	17
	4.1 RESULT	17
	4.2 ANALYSIS	17
5.	SUMMARY AND CONCLUSIONS	18
	REFERENCES	18
	APPENDIX	18
	A. SOURCE CODE	36
	B. SCREENSHOTS	18

LIST OF FIGURES

FIGURE No.	TITLE	PAGE No.
1.1	TYPES OF LEARNING	2
1.2	SUPERVISED LEARNING	3
1.3	CLASSIFICATION	4
1.4	LOGISTIC REGRESSION	5
1.6	BAR GRAPH	11
1.7	BAR GRAPH 2	11
1.8	HEAT MAP	12
1.9	DATASET	14
2.2	REPORT	15
2.3	RESULT	17
2.4	ANALYSIS	17

LIST OF TABLES

TABLE No.	TITLE	PAGE No.
1.5	DATASET	7

LIST OF ABBREVIATIONS

ML	-	Machine Learning
AI	-	Artificial Intelligence
MLE	-	Maximum Likelihood Estimation
SGD	-	Stochastic Gradient Descent
LR	-	Logistic Regression

CHAPTER-1

INTRODUCTION

What is Machine Learning?

Machine learning (ML) is a type of artificial intelligence (AI) that allows software applications to become more accurate at predicting outcomes without being explicitly programmed to do so. Machine learning algorithms use historical data as input to predict new output values. Recommendation are a common use case for machine learning. Other popular uses include fraud detection, spam filtering, malware threat detection, business automation (BPA) and predictive maintenance.

Why is machine learning important?

Machine learning is important because it gives enterprises a view of trends in customer behavior and business operational patterns, as well as supports the development of new products. Many of today's leading companies, such as Facebook, Google and Uber, make machine learning a central part of their operations. Machine learning has become a significant competitive differentiator for many companies.

Machine Learning History

- A machine that is intellectually capable as much as humans has always fired the imagination of writers and also the early computer scientist who were excited about artificial intelligence and machine learning, but the first machine learning system was developed in the 1950s.
- In 1952, Arthur Samuel was at IBM. He developed a program for playing Checkers. The program was able to observe positions at the game and learn a model that gives better moves for the machine player. The system played many games with the program and observed that the program was able to play better in the course of time with getting more experience of board games. Samuel coined the term machine learning and he defined learning as a field of study that gives computers the ability without being explicitly programmed.
- In 1957, Rosenblatt proposed the perceptron. Perceptron is the simple neural network unit; it was a very exciting discovery at that time. Rosenblatt made the following statement; the perceptron is designed to illustrate some of the fundamental properties of intelligent systems in general without becoming too deeply immersed in the special and frequently unknown conditions, which hold force particular biological organisms.
- But after 3 years, came up with the delta learning rule that is used for learning perceptron. It was used as a procedure for training perceptron. It is also known as the least square problem. The combination of these ideas created a good linear classifier. However, the work along these lines suffered a setback when Minsky in 1969 came up with the limitations of perceptron. He showed, that the problem could not be represented by perceptron and such inseparable data distributions cannot be handled and following

What are the different types of ML?

Classical machine learning is often categorized by how an algorithm learns to become more accurate in its predictions. There are four basic approaches: supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning. The type of algorithm data scientists choose to use depends on what type of data they want to predict.

- **Supervised learning:** In this type of machine learning, data scientists supply algorithms with labeled training data and define the variables they want the algorithm to assess for correlations. Both the input and the output of the algorithm is specified.
- **Unsupervised learning:** This type of machine learning involves algorithms that train on unlabeled data. The algorithm scans through data sets looking for any meaningful connection. The data that algorithms train on as well as the predictions or recommendations they output are predetermined.
- **Semi-supervised learning:** This approach to machine learning involves a mix of the two preceding types. Data scientists may feed an algorithm mostly labeled training data, but the model is free to explore the data on its own and develop its own understanding of the data set.
- **Reinforcement learning:** Data scientists typically use reinforcement learning to teach a machine to complete a multi-step process for which there are clearly defined rules. Data scientists program an algorithm to complete a task and give it positive or negative cues as it works out how to complete a task. But for the most part, the algorithm decides on its own what steps to take along the way.

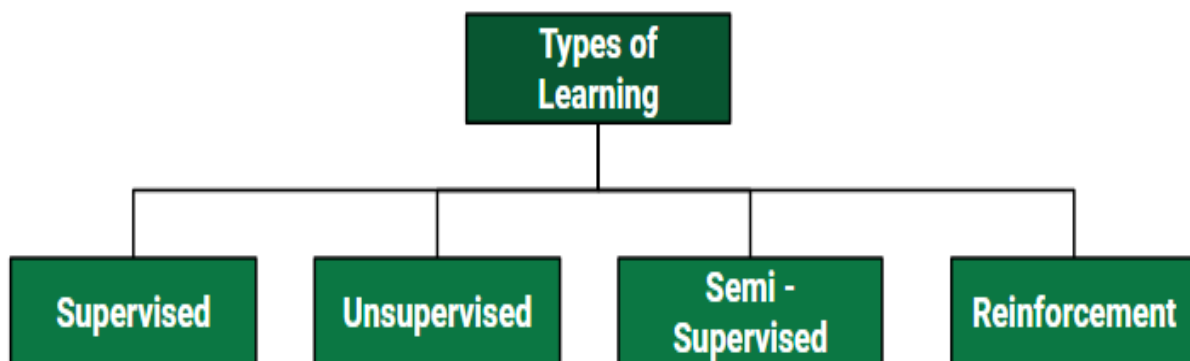


Fig 1.1: Types of Learning

SUPERVISED MACHINE LEARNING:

In supervised learning, models are trained using labelled dataset, where the model learns about each type of data. Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.

Types of Supervised Machine Learning Algorithm:

Supervised learning can be further divided into two types of problems:

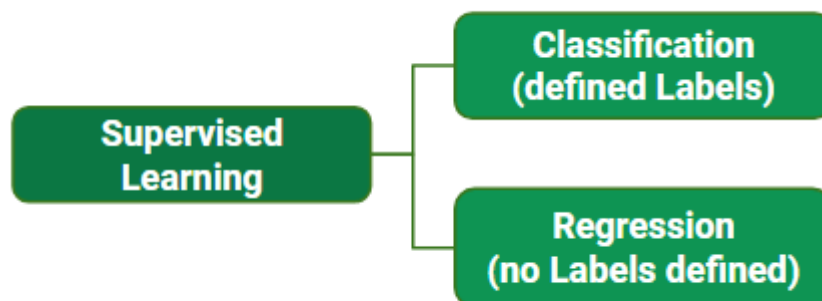


FIG 1.2: Supervised

Classification

The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data. In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups. Such

as, **Yes or No, 0 or 1, Spam or Not Spam, cat or dog**, etc. Classes can be called as targets/labels or categories.

In classification algorithm, a discrete output function(y) is mapped to input variable(x).

The best example of an ML classification algorithm is **Email Spam Detector**.

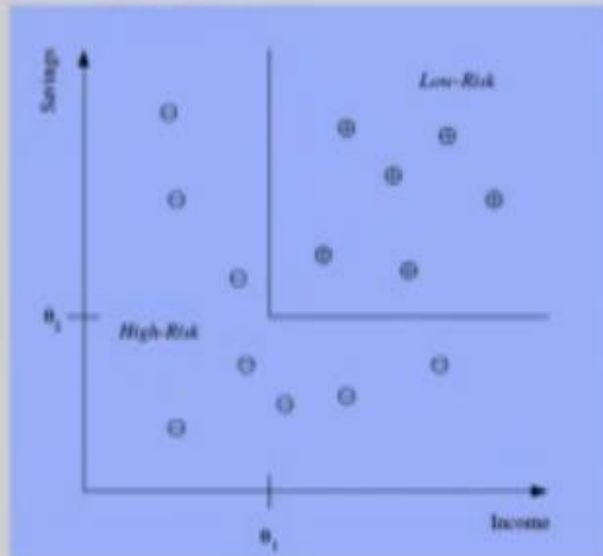
The main goal of the Classification algorithm is to identify the category of a given dataset, and these algorithms are mainly used to predict the output for the categorical data.

In the below diagram, there are two classes, class A and Class B. These classes have features that are similar to each other and dissimilar to other classes.

Classification

Example: Credit scoring

Differentiating between **low-risk** and **high-risk** customers from their *income* and *savings*



Discriminant: IF *income* > θ_1 AND *savings* > θ_2
THEN **low-risk** ELSE **high-risk**

FIG 1.3: CLASSIFICATION

Logistic Regression

Logistic Regression is a Machine Learning algorithm which is used for the classification problems, it is a predictive analysis algorithm and based on the concept of probability.

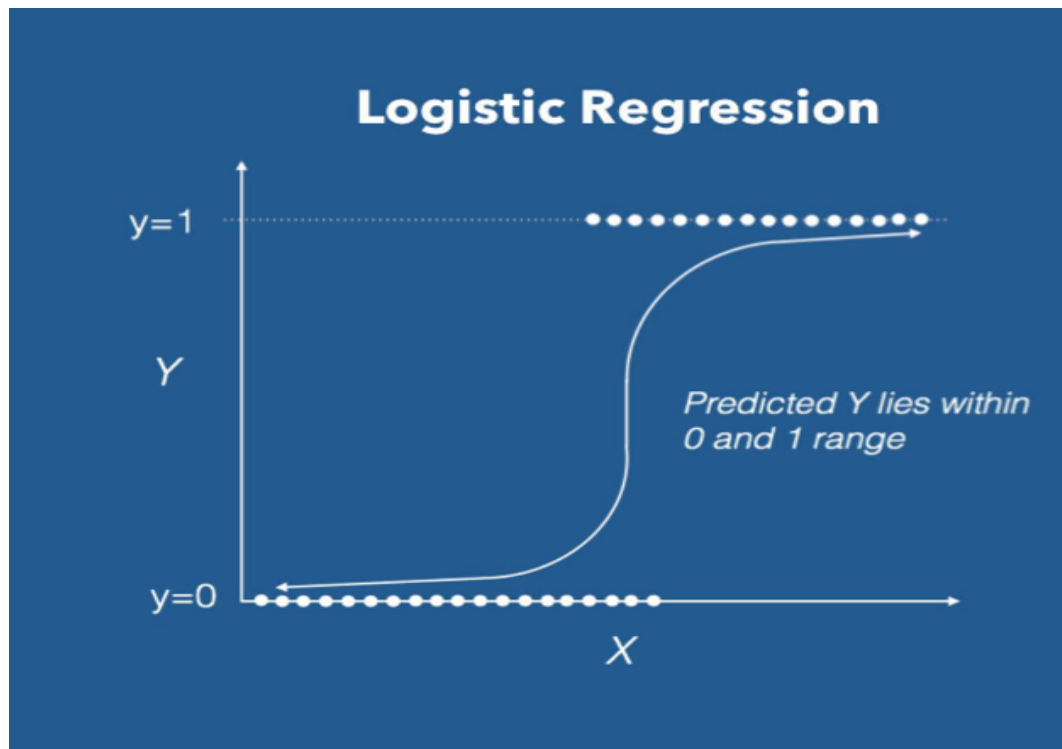


FIG 1.4: LOGISTIC

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Some of the examples of classification problems are Email spam or not spam, Online transactions Fraud or not Fraud, Tumour Malignant or Benign. Logistic regression transforms its output using the logistic sigmoid function to return a probability value.

What are the types of logistic regression?

1. Binary (e.g. Tumor: Malignant or Benign)
2. Multi-linear functions fails Class (e.g. Cats, dogs or Sheep's)

CHAPTER-2

AIM AND SCOPE OF THE PRESENT INVESTIGATION:

AIM:

The aim of the project is to predict how likely it is that the people will take an H1N1 flu vaccine using Logistic Regression. Training logistic regression model with MLE and SGD for prediction and finding out which model get the highest accuracy is the scope of the project.

SCOPE:

- Training logistic regression model with Maximum likelihood estimation and Stochastic gradient descent for prediction
- Finding out which model gets the highest accuracy is the scope of the project.

CHAPTER-3

EXPERIMENTAL OR MATERIALS AND METHODS, ALGORITHMS USED

VACCINE USAGE PREDICTION

Overview:

- The aim of the project is to predict how likely it is that the people will take an H1N1 flu vaccine using Logistic Regression.
- Training logistic regression model with MLE and SGD for prediction is the scope of the project.
- Supervised Machine learning has been used to predict the accuracy rate.

Requirements:

To build the machine learning model, the main thing needed is the datasets .The dataset that are used for the prediction are listed below:

DATASET INFORMATION:

Column	Description
unique_id	Unique identifier for each respondent
h1n1_worry	Worry about the h1n1 flu(0,1,2,3) 0=Not worried at all, 1=Not very worried, 2=Somewhat worried, 3=Very worried
h1n1_awareness	Signifies the amount of knowledge or understanding the respondent has about h1n1 flu - (0,1,2) - 0=No knowledge, 1=little knowledge, 2=good knowledge
antiviral_medication	Has the respondent taken antiviral vaccination - (0,1)
contact_avoidance	Has avoided any close contact with people who have flu-like symptoms - (0,1)

bought_face_mask	Has the respondent bought mask or not - (0,1)
wash_hands_frequently	Washes hands frequently or uses hand sanitizer - (0,1)
avoid_large_gatherings	Has the respondent reduced time spent at large gatherings - (0,1)
reduced_outside_home_cont	Has the respondent reduced contact with people outside their own house - (0,1)
avoid_touch_face	Avoids touching nose, eyes, mouth - (0,1)
dr_recc_h1n1_vacc	Doctor has recommended h1n1 vaccine - (0,1)
dr_recc_seasonal_vacc	The doctor has recommended seasonal flu vaccine - (0,1)
chronic_medical_condition	Has any chronic medical condition - (0,1)
cont_child_under_6_mnth	Has regular contact with child the age of 6 months - (0,1)
is_health_worker	Is respondent a health worker - (0,1)
has_health_insurance	Does respondent have health insurance - (0,1)
is_h1n1_vaccine_effective	Does respondent think that the h1n1 vaccine is effective - (1,2,3,4,5)- (1=Thinks not effective at all, 2=Thinks it is not very effective, 3=Doesn't know if it is effective or not, 4=Thinks it is somewhat effective, 5=Thinks it is highly effective)
is_h1n1_risky	What respondents think about the risk of getting ill with h1n1 in the absence of the vaccine- (1,2,3,4,5)- (1=Thinks it is not very low risk, 2=Thinks it is somewhat low risk, 3=don't know if it is risky or not, 4=Thinks it is a somewhat high risk, 5=Thinks it is very highly risky)

sick_from_h1n1_vacc	Does respondent worry about getting sick by taking the h1n1 vaccine - (1,2,3,4,5)- (1=Respondent not worried at all, 2=Respondent is not very worried, 3=Doesn't know, 4=Respondent is somewhat worried, 5Respondent is very worried) -
is_seas_vacc_effective	Does respondent think that the seasonal vaccine is effective- (1,2,3,4,5)- (1=Thinks not effective at all, 2=Thinks it is not very effective, 3=Doesn't know if it
	is effective or not, 4=Thinks it is somewhat effective, 5=Thinks it is highly effective)
is_seas_flu_risky	What respondenst think about the risk of getting ill with seasonal flu in the absence of the vaccine- (1,2,3,4,5)- (1=Thinks it is not very low risk, 2=Thinks it is somewhat low risk, 3=Doesn't know if it is risky or not, 4=Thinks it is somewhat high risk, 5=Thinks it is very highly risky)
sick_from_seas_vacc	Does respondent worry about getting sick by taking the seasonal flu vaccine - (1,2,3,4,5)- (1=Respondent not worried at all, 2=Respondent is not very worried, 3=Doesn't know, 4=Respondent is somewhat worried, 5Respondent is very worried)
age_bracket	Age bracket of the respondent - (18 - 34 Years, 35 - 44 Years, 45 - 54 Years, 55 - 64 Years, 64+ Years)
qualification	Qualification/education level of the respondent as per their response -(<12 Years, 12 Years, College Graduate, Some College)
race	Respondent's race - (White, Black, Other or Multiple ,Hispanic)
sex	Respondent's sex - (Female, Male)
income_level	Annual income of the respondent as per the 2008 poverty Census - (<= 75000–AbovePoverty,> 75000–AbovePoverty,>75000, Below Poverty)
marital_status	Respondent's marital status - (Not Married, Married)

housing_status	Respondent's housing status - (Own, Rent)
employment	Respondent's employment status - (Not in Labor Force, Employed, Unemployed)
census_msa	Residence of the respondent with the MSA(metropolitan statistical area)(Non-MSA, MSA- Not Principle, CityMSA- Principle city) - (Yes, no)
no_of_adults	Number of adults in the respondent's house (0,1,2,3) - (Yes, no)
no_of_children	Number of children in the respondent's house(0,1,2,3) - (Yes, No)
h1n1_vaccine	Dependent variable)Did the respondent received the h1n1 vaccine or not(1,0) - (Yes, No)

TABLE 1.5: DATASET

METHODOLOGY:

- **IMPORTING ALL THE NECESSARY LIBRARIES:**

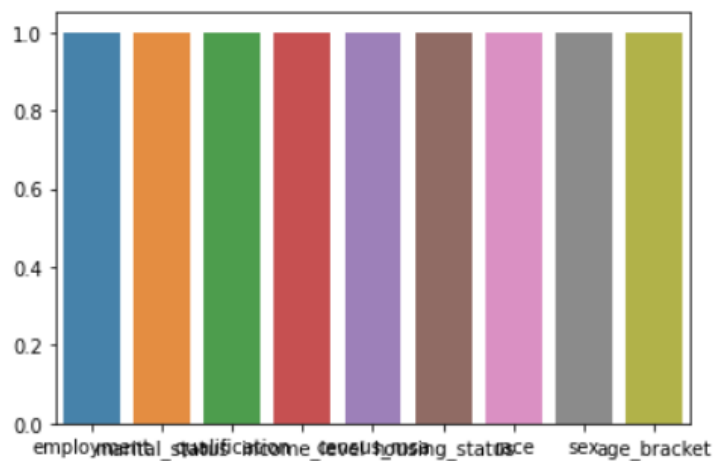
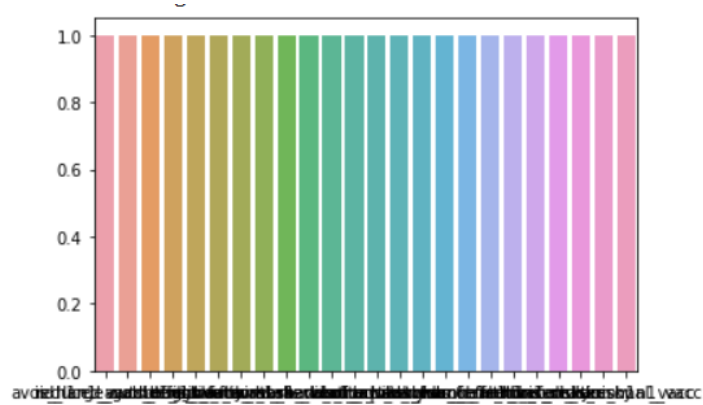
1. NUMPY
2. MATPLOTLIB
3. SEABORN
4. PANDAS
5. SKLEARN

- **DATA ANALYSIS:**

Data analysis is the first step to do the project because data exploration is very important to know the inputs. Data Analysis is the process of inspecting, cleansing, transforming, and modelling data with the goal of discovering useful information by informing conclusions and supporting decision making.

1. **GRAPHS:**

They are used to represent, find, analyse, and optimize connections between elements.
Ex: Bar graph, Heat map



- Bar graphs are used to compare things between different groups or to track changes over time.

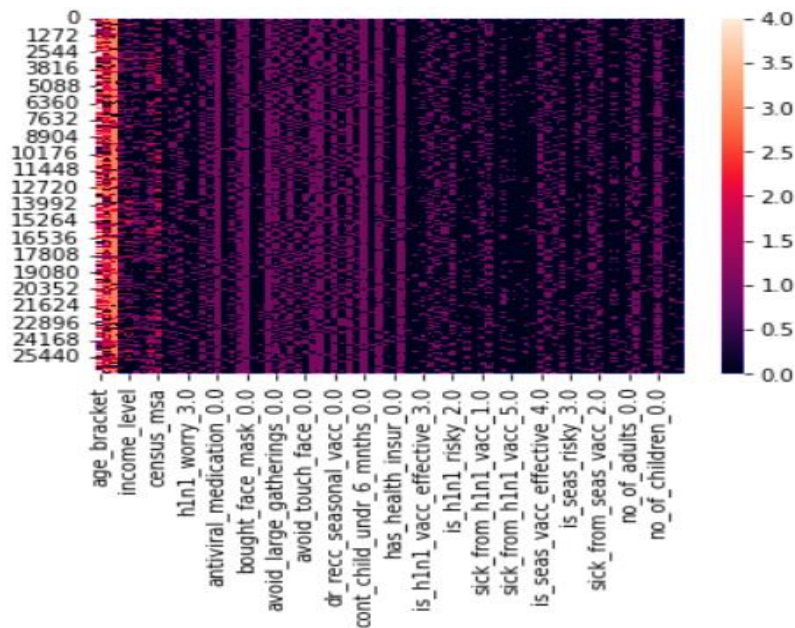


FIG 1.8: HEAT MAP

- Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix

2. DATA CLEANING:

Removing null records, dropping unnecessary columns, treating missing values, rectifying junk values or otherwise called outliers, restructuring the data to modify it to a more readable format, etc is known as data cleaning.

• DATA PREPROCESSING:

Data pre-processing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.

When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data pre-processing task.

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data pre-processing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

1. Changing the datatype:

Dividing the data into categorical and numerical data subsets and then converting it into categorical data and then into numerical data. so as to convert them into the machine-readable form. Machine learning algorithms can then decide in a better way how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning. Then merge the data subsets into one dataset.

2. Extracting dependent and independent variables:

In machine learning, it is important to distinguish the matrix of features (independent variables) and dependent variables from dataset. In our dataset, there are independent variables that are no of adults, no of children and one is a dependent variable which is h1n1 vaccine.

- **Extracting independent variable:**

To extract an independent variable, we will use `iloc[ ]` method of Pandas library. It is used to extract the required rows and columns from the dataset.

```
X= data_set.iloc[:, :-1].values
```

In the above code, the first colon(:) is used to take all the rows, and the second colon(:) is for all the columns. Here we have used `:-1`, because we don't want to take the last column as it contains the dependent variable. So by doing this, we will get the matrix of features.

- **Extracting dependent variable:**

To extract dependent variables, again, we will use Pandas `.iloc[]` method

```
y= data_set.iloc[:, 3].values
```

Here we have taken all the rows with the last column only. It will give the array of dependent variables.

3. Dummy Variables:

Dummy variables are those variables which have values 0 or 1. The 1 value gives the presence of that variable in a particular column, and rest variables become 0. With dummy encoding, we will have a number of columns equal to the number of categories.

In our dataset, we have 3 categories so it will produce three columns having 0 and 1 values. For Dummy Encoding, we will use **OneHotEncoder** class of **pre-processing** library.

Splitting the Dataset into the Training set and Test set

In machine learning data preprocessing, we divide our dataset into a training set and test set. This is one of the crucial steps of data preprocessing as by doing this, we can enhance the performance of our machine learning model.

Suppose, if we have given training to our machine learning model by a dataset and we test it by a completely different dataset. Then, it will create difficulties for our model to understand the correlations between the models.

If we train our model very well and its training accuracy is also very high, but we provide a new dataset to it, then it will decrease the performance. So we always try to make a machine learning model which performs well with the training set and also with the test dataset. Here, we can define these datasets as:

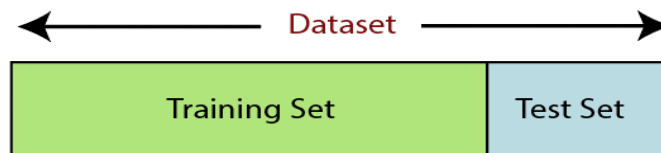


FIG 1.9: DATASET

Training Set: A subset of dataset to train the machine learning model, and we already know the output.

Test set: A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

For splitting the dataset, we will use the below lines of code:

1. `from sklearn.model_selection import train_test_split`
2. `Xtrain, Xtest, ytrain, ytest= train_test_split(X, y, test_size= 0.2)`

Explanation:

In the above code, the first line is used for splitting arrays of the dataset into random train and test subsets.

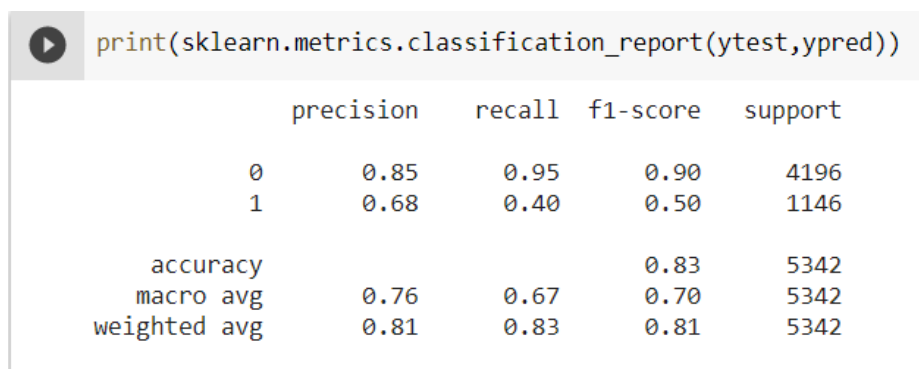
- **Xtrain:** features for the training data
- **Xtest:** features for testing data
- **ytrain:** Dependent variables for training data
- **ytest:** Independent variable for testing data
- In **train_test_split()** function, we have passed four parameters.

which first two are for arrays of data, and **test_size** is for specifying the size of the test set. The test_size maybe .5, .3, or .2, which tells the dividing ratio of training and testing sets

4. Analyzing Several Models on the Dataset:

- ❖ Import logistic regression from sklearn library and train the model and predicted the class labels for Xtest and ytest.
- ❖ Then from sklearn.metrics we have to import:
 - **CLASSIFICATION_REPORT**: It gives the precision, recall, f1 score and support from the ytest and ypred.

EX.



```
print(sklearn.metrics.classification_report(ytest,ypred))
```

	precision	recall	f1-score	support
0	0.85	0.95	0.90	4196
1	0.68	0.40	0.50	1146
accuracy			0.83	5342
macro avg	0.76	0.67	0.70	5342
weighted avg	0.81	0.83	0.81	5342

FIG 2.1: REPORT

- **CONFUSION_MATRIX**: to evaluate the accuracy of a classification.
- **ACCURACY_SCORE**: In multilabel classification, this function computes subset accuracy: the set of labels predicted for a sample must *exactly* match the corresponding set of labels in y_test.
- Then find the default logistic regression accuracy score of the train and test set.
- Then by logistic regression using:
 - **Maximum likelihood Estimation (MLE)**: is a frequentist approach for estimating the parameters of a model given some observed data.
 - **Stochastic gradient descent (SGD)**: is an optimization algorithm often used in machine learning applications to find the model parameters that correspond to the best fit between predicted and actual outputs.

5. COMPARE THE VARIOUS MODELS:

- Build the models.
- Spot check the algorithms.
- Evaluate each model in turn.
- Make predictions on the validation set.
- At last find out which model has the highest accuracy rate in prediction.

CHAPTER- 4

RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS

Result:

I have predicted by logistic regression that 4651 people have not vaccinated and 691 people have got vaccinated.

```
[ ] Vaccinators = pd.DataFrame(ypred)    #0-has taken ,1-has not taken
    Vaccinators.value_counts()

0    4649
1     693
dtype: int64
```

FIG 2.2: RESULT

ANALYSIS:

By comparing various other models, I have found out that logistic regression and support vector machine gave the highest accuracy rate during the prediction.

```
https://scikit-learn.org/stable/modules/linear\_model.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
LR: 0.834635 (0.006828)
LDA: 0.831920 (0.007474)
KNN: 0.805476 (0.006011)
CART: 0.747999 (0.005492)
NB: 0.690990 (0.011495)
SVM: 0.833746 (0.005830)
DecisionTree: 0.751182 (0.007319)
Random Forest: 0.833044 (0.007227)
```

FIG 2.3: ANALYSIS

CHAPTER- 5

SUMMARY AND CONCLUSION

So, by using logistic regression and various other machine learning models we can predict how likely the person will take an H1N1 flu vaccine. Training logistic regression model with MLE and SGD for the prediction. Various datasets have been taken as the inputs to get the most accuracy rate. I got the highest accuracy rate in logistic regression and random forest models.

REFERENCES:

Panda, N.; Majhi, S.K. How effective is the salp swarm algorithm in data classification. In Computational Intelligence in Pattern Recognition; Springer: Singapore, 2020; pp. 579–588.

APPENDIX

A. SCREENSHOTS

TOPIC: Predict how likely it is that the people will take an H1N1 flu vaccine using Logistic Regression

IMPORTING ALL THE NECESSARY LIBRARIES FOR VACCINE PREDICTION

```
[ ] import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
```

```
[ ] data = pd.read_csv('/content/h1n1_vaccine_prediction.csv')
```

data.dtypes

unique_id	int64
h1n1_worry	float64
h1n1_awareness	float64
antiviral_medication	float64
contact_avoidance	float64
bought_face_mask	float64
wash_hands_frequently	float64
avoid_large_gatherings	float64
reduced_outside_home_cont	float64
avoid_touch_face	float64
dr_recc_h1n1_vacc	float64
dr_recc_seasonal_vacc	float64
chronic_medication	float64
cont_child_under_6_mnths	float64
is_health_worker	float64
has_health_insurance	float64
is_h1n1_vacc_effective	float64
is_h1n1_risky	float64
sick_from_h1n1_vacc	float64
is_seasonal_vacc_effective	float64
is_seasonal_risky	float64
sick_from_seasonal_vacc	float64
age_bracket	object
qualification	object
race	object
sex	object
income_level	object
marital_status	object
housing_status	object
employment	object
census_msa	object

```
housing_status      object
employment          object
census_msa          object
no_of_adults        float64
no_of_children       float64
h1n1_vaccine        int64
dtype: object
```

```
[ ] data = data.drop(columns=['unique_id'])
```

```
[ ] data.columns
```

```
Index(['h1n1_worry', 'h1n1_awareness', 'antiviral_medication',
       'contact_avoidance', 'bought_face_mask', 'wash_hands_frequently',
       'avoid_large_gatherings', 'reduced_outside_home_cont',
       'avoid_touch_face', 'dr_recc_h1n1_vacc', 'dr_recc_seasonal_vacc',
       'chronic_medication', 'cont_child_undr_6_mnths',
       'is_health_worker', 'has_health_insur', 'is_h1n1_vacc_effective',
       'is_h1n1_risky', 'sick_from_h1n1_vacc', 'is_seas_vacc_effective',
       'is_seas_risky', 'sick_from_seas_vacc', 'age_bracket', 'qualification',
       'race', 'sex', 'income_level', 'marital_status', 'housing_status',
       'employment', 'census_msa', 'no_of_adults', 'no_of_children',
       'h1n1_vaccine'],
      dtype='object')
```

```
[ ] data_category_column = data.columns[data.dtypes=='object'] #which of the columns are category type
```

```
[ ] data_numeric_column = data.columns[data.dtypes!='object'] #which of the columns are of numeric type
```

```
[ ] print(data_category_column) #Names of category columns
```

```
Index(['age_bracket', 'qualification', 'race', 'sex', 'income_level',
       'marital_status', 'housing_status', 'employment', 'census_msa'],
      dtype='object')
```

```
[ ] print(data_numeric_column) #Names of numeric columns
```

```
Index(['h1n1_worry', 'h1n1_awareness', 'antiviral_medication',
       'contact_avoidance', 'bought_face_mask', 'wash_hands_frequently',
       'avoid_large_gatherings', 'reduced_outside_home_cont',
       'avoid_touch_face', 'dr_recc_h1n1_vacc', 'dr_recc_seasonal_vacc',
       'chronic_medication', 'cont_child_undr_6_mnths',
       'is_health_worker', 'has_health_insur', 'is_h1n1_vacc_effective',
       'is_h1n1_risky', 'sick_from_h1n1_vacc', 'is_seas_vacc_effective',
       'is_seas_risky', 'sick_from_seas_vacc', 'no_of_adults',
       'no_of_children', 'h1n1_vaccine'],
      dtype='object')
```

Create data subsets

```
data_category = data[data_category_column] #Category data
```

+ Code

+ Text

```
[ ] data_numeric = data[data_numeric_column] #Numeric data
```

```
[ ] data_category.shape
```

```
(26707, 9)
```

```
[ ] data_numeric.shape
```

```
(26707, 24)
```

```
[ ] data_category.head()
```

	age_bracket	qualification	race	sex	income_level	marital_status	housing_status	employment	census_msa
0	55 - 64 Years	< 12 Years	White	Female	Below Poverty	Not Married	Own	Not in Labor Force	Non-MSA
1	35 - 44 Years	12 Years	White	Male	Below Poverty	Not Married	Rent	Employed	MSA, Not Principle City
2	18 - 34 Years	College Graduate	White	Male	<= \$75,000, Above Poverty	Not Married	Own	Employed	MSA, Not Principle City
3	65+ Years	12 Years	White	Female	Below Poverty	Not Married	Rent	Not in Labor Force	MSA, Principle City
4	45 - 54 Years	Some College	White	Female	<= \$75,000, Above Poverty	Married	Own	Employed	MSA, Not Principle City

```
[ ] data_numeric.head()
```

	h1n1_worry	h1n1_awareness	antiviral_medication	contact_avoidance	bought_face_mask	wash_hands_frequently	avoid_large_gatherings	reduced_outside_home_cont
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1	3.0	2.0	0.0	1.0	0.0	1.0	0.0	1.0
2	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0
3	1.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0
4	2.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0

Check for missing data

```
[ ] data_category.isna().sum().sort_values(ascending=False)
```

```
income_level      4423
housing_status    2042
employment         1463
marital_status    1408
qualification      1407
census_msa         0
sex               0
race              0
age_bracket       0
dtype: int64
```

```
[ ] data_numeric.isna().sum().sort_values(ascending=False)
```

```
has_health_insur      12274
dr_recc_h1n1_vacc      2160
dr_recc_seasonal_vacc  2160
chronic_medic_condition  971
cont_child_unchr_6_mnth  820
is_health_worker       804
sick_from_seas_vacc     537
is_seas_risky          514
is_seas_vacc_effective  462
sick_from_h1n1_vacc     395
is_h1n1_vacc_effective  388
is_h1n1_risky          388
no_of_children         249
no_of_adults           249
contact_avoidance      208
avoid_touch_face       128
h1n1_awareness         116
h1n1_worry             92
avoid_large_gatherings  87
reduced_outside_home_cont  82
antiviral_medication    71
wash_hands_frequently   42
bought_face_mask        19
h1n1_vaccine            0
dtype: int64
```

Categorical imputation using `lambda x:x.fillna(x.value_counts().index[0])`

```
[ ] data_clean_category = data_category.apply(lambda x: x.fillna(x.value_counts().index[0]))
```

```
[ ] data_clean_category = data_category.apply(lambda x: x.fillna(x.value_counts().index[0]))
```

```
[ ] data_clean_category.isna().sum().sort_values(ascending=False)
```

```
census_msa      0
employment      0
housing_status  0
marital_status  0
income_level    0
sex             0
race            0
qualification    0
age_bracket     0
dtype: int64
```

```
[ ] data_clean_category.dtypes
```

```
age_bracket      object
qualification     object
race             object
sex              object
income_level     object
marital_status   object
housing_status   object
employment       object
census_msa       object
dtype: object
```

use a loop to convert all object columns to category

```
[ ] for column in data_clean_category.columns:
    data_clean_category[column] = data_clean_category[column].astype('category')
```

```
[ ] data_clean_category.dtypes
```

```
age_bracket      category
qualification     category
race             category
sex              category
income_level     category
marital_status   category
housing_status   category
employment       category
census_msa       category
dtype: object
```

```
[ ] data_clean_category.head()
```

 data_clean_category.head()

	age_bracket	qualification	race	sex	income_level	marital_status	housing_status	employment	census_msa
0	55 - 64 Years	< 12 Years	White	Female	Below Poverty	Not Married	Own	Not in Labor Force	Non-MSA
1	35 - 44 Years	12 Years	White	Male	Below Poverty	Not Married	Rent	Employed	MSA, Not Principle City
2	18 - 34 Years	College Graduate	White	Male	<= \$75,000, Above Poverty	Not Married	Own	Employed	MSA, Not Principle City
3	65+ Years	12 Years	White	Female	Below Poverty	Not Married	Rent	Not in Labor Force	MSA, Principle City
4	45 - 54 Years	Some College	White	Female	<= \$75,000, Above Poverty	Married	Own	Employed	MSA, Not Principle City

 Code  Text

Numeric columns missing value imputation

[] data_numeric.head()

	h1n1_worry	h1n1_awareness	antiviral_medication	contact_avoidance	bought_face_mask	wash_hands_frequently	avoid_large_gatherings	reduced_outside_home_cont
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1	3.0	2.0	0.0	1.0	0.0	1.0	0.0	1.0
2	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0
3	1.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0
4	2.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0

[] data_numeric.dtypes

h1n1_worry	float64
h1n1_awareness	float64
antiviral_medication	float64
contact_avoidance	float64
bought_face_mask	float64
wash_hands_frequently	float64
avoid_large_gatherings	float64
reduced_outside_home_cont	float64
avoid_touch_face	float64
dr_recc_h1n1_vacc	float64
dr_recc_seasonal_vacc	float64
chronic_medication	float64
cont_child_unclr_6_mnth	float64
is_health_worker	float64
has_health_insur	float64
is_h1n1_vacc_effective	float64

```
is_seas_vacc_effective    float64
is_seas_risky             float64
sick_from_seas_vacc       float64
no_of_adults              float64
no_of_children            float64
h1n1_vaccine              int64
dtype: object
```

```
[ ] data_clean_numeric = data_numeric.apply(lambda x: x.fillna(x.value_counts().index[0]))
```

```
[ ] data_clean_numeric.head()
```

```
h1n1_worry    h1n1_awareness    antiviral_medication    contact_avoidance    bought_face_mask    wash_hands_frequently    avoid_large_gathering
0            1.0            0.0            0.0            0.0            0.0            0.0            0
1            3.0            2.0            0.0            1.0            0.0            1.0            0
2            1.0            1.0            0.0            1.0            0.0            0.0            0
3            1.0            1.0            0.0            1.0            0.0            1.0            1
4            2.0            1.0            0.0            1.0            0.0            1.0            1
```

```
[ ] data_clean_numeric.isna().sum()
```

```
h1n1_worry            0
h1n1_awareness        0
antiviral_medication  0
contact_avoidance     0
bought_face_mask      0
wash_hands_frequently 0
avoid_large_gatherings 0
reduced_outside_home_cont 0
avoid_touch_face      0
dr_recc_h1n1_vacc     0
dr_recc_seasonal_vacc 0
chronic_med_condition 0
cont_child_undr_6_mnths 0
is_health_worker      0
has_health_insur      0
is_h1n1_vacc_effective 0
is_h1n1_risky         0
sick_from_h1n1_vacc   0
is_seas_vacc_effective 0
is_seas_risky         0
sick_from_seas_vacc   0
no_of_adults          0
no_of_children         0
h1n1_vaccine          0
dtype: int64
```

```
no_of_children    0
h1n1_vaccine      0
dtype: int64
```

```
data_clean_numeric.dtypes
```

```
h1n1_worry            float64
h1n1_awareness        float64
antiviral_medication  float64
contact_avoidance     float64
bought_face_mask      float64
wash_hands_frequently float64
avoid_large_gatherings float64
reduced_outside_home_cont float64
avoid_touch_face      float64
dr_recc_h1n1_vacc     float64
dr_recc_seasonal_vacc float64
chronic_med_condition float64
cont_child_undr_6_mnths float64
is_health_worker      float64
has_health_insur      float64
is_h1n1_vacc_effective float64
is_h1n1_risky         float64
sick_from_h1n1_vacc   float64
is_seas_vacc_effective float64
is_seas_risky         float64
sick_from_seas_vacc   float64
no_of_adults          float64
no_of_children         float64
h1n1_vaccine          int64
dtype: object
```

```
[ ] data_clean_numeric = data_clean_numeric.astype('category')
```

```
[ ] data_clean_numeric.head()
```

```
h1n1_worry    h1n1_awareness    antiviral_medication    contact_avoidance    bought_face_mask    wash_hands_frequer
0            1.0            0.0            0.0            0.0            0.0
1            3.0            2.0            0.0            1.0            0.0
2            1.0            1.0            0.0            1.0            0.0
3            1.0            1.0            0.0            1.0            0.0
4            2.0            1.0            0.0            1.0            0.0
```

```
[ ] data_clean_numeric['h1n1_worry'].value_counts()
```

```
2.0    10667
```

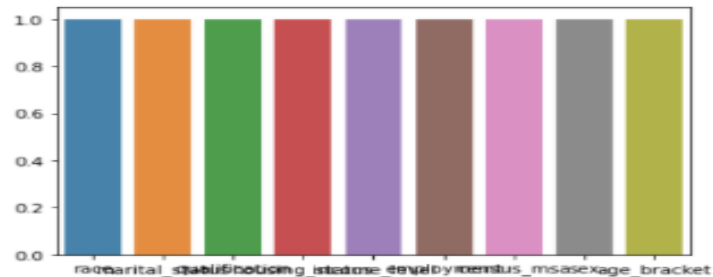
```
data_clean_numeric['h1n1_worry'].value_counts()
```

```
2.0    10667
1.0     8153
3.0     4591
0.0     3296
Name: h1n1_worry, dtype: int64
```

```
[ ] category_count = data_clean_category.columns.value_counts()
```

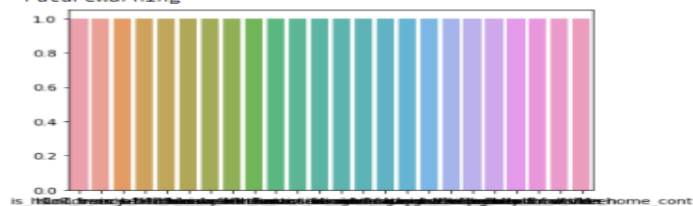
```
[ ] sns.barplot(category_count.index,category_count.values,alpha=0.9)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning:
FutureWarning
```



```
sns.barplot(numeric_count.index,numeric_count.values,alpha=0.9)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the
FutureWarning
```



```
[ ] data_clean_category.columns
```

```
Index(['age_bracket', 'qualification', 'race', 'sex', 'income_level',
       'marital_status', 'housing_status', 'employment', 'census_msa'],
      dtype='object')
```

```
[ ] data_clean_numeric.columns
```

```
Index(['h1n1_worry', 'h1n1_awareness', 'antiviral_medication',
       'contact_avoidance', 'bought_face_mask', 'wash_hands_frequently',
       'avoid_large_gatherings', 'reduced_outside_home_cont',
       'avoid_touch_face', 'dr_recc_h1n1_vacc', 'dr_recc_seasonal_vacc',
       'chronic_medical_condition', 'cont_child_unchr_6_mnths',
       'is_health_worker', 'has_health_insur', 'is_h1n1_vacc_effective',
       'is_h1n1_risky', 'sick_from_h1n1_vacc', 'is_seas_vacc_effective',
       'is_seas_risky', 'sick_from_seas_vacc', 'no_of_adults',
       'no_of_children', 'h1n1_vaccine'],
      dtype='object')
```

```
[ ] data_clean_category.dtypes
```

```
age_bracket          category
qualification        category
race                 category
sex                  category
income_level         category
marital_status       category
housing_status       category
```

census_msa	category
dtype: object	

data_clean_numeric.dtypes	
h1n1_worry	category
h1n1_awareness	category
antiviral_medication	category
contact_avoidance	category
bought_face_mask	category
wash_hands_frequently	category
avoid_large_gatherings	category
reduced_outside_home_cont	category
avoid_touch_face	category
dr_recc_h1n1_vacc	category
dr_recc_seasonal_vacc	category
chronic_medical_condition	category
cont_child_unchr_6_mnth	category
is_health_worker	category
has_health_insur	category
is_h1n1_vacc_effective	category
is_h1n1_risky	category
sick_from_h1n1_vacc	category
is_seas_vacc_effective	category
is_seas_risky	category
sick_from_seas_vacc	category
no_of_adults	category
no_of_children	category
h1n1_vaccine	category
dtype: object	

data_clean_category										
	age_bracket	qualification	race	sex	income_level	marital_status	housing_status	employment	census_msa	
0	55 - 64 Years	< 12 Years	White	Female	Below Poverty	Not Married	Own	Not in Labor Force	Non-MSA	
1	35 - 44 Years	12 Years	White	Male	Below Poverty	Not Married	Rent	Employed	MSA, Not Principle City	
2	18 - 34 Years	College Graduate	White	Male	<= \$75,000, Above Poverty	Not Married	Own	Employed	MSA, Not Principle City	
3	65+ Years	12 Years	White	Female	Below Poverty	Not Married	Rent	Not in Labor Force	MSA, Principle City	
4	45 - 54 Years	Some College	White	Female	<= \$75,000, Above Poverty	Married	Own	Employed	MSA, Not Principle City	
...	...	...	...	...	...	...	...	...	...	
26702	65+ Years	Some College	White	Female	<= \$75,000, Above Poverty	Not Married	Own	Not in Labor Force	Non-MSA	
26703	18 - 34 Years	College Graduate	White	Male	<= \$75,000, Above Poverty	Not Married	Rent	Employed	MSA, Principle City	
26704	55 - 64 Years	Some College	White	Female	<= \$75,000, Above Poverty	Not Married	Own	Employed	MSA, Not Principle City	
26705	18 - 34 Years	Some College	Hispanic	Female	<= \$75,000, Above Poverty	Married	Rent	Employed	Non-MSA	

26705	18 - 34 Years	Some College	Hispanic	Female	<= \$75,000, Above Poverty	Married	Rent	Employed	Non-MSA	
26706	65+ Years	Some College	White	Male	<= \$75,000, Above Poverty	Married	Own	Not in Labor Force	MSA, Principle City	

26707 rows x 9 columns


```

data_clean_category['age_bracket'] = data_clean_category['age_bracket'].cat.codes
data_clean_category['qualification'] = data_clean_category['qualification'].cat.codes
data_clean_category['race'] = data_clean_category['race'].cat.codes
data_clean_category['sex'] = data_clean_category['sex'].cat.codes
data_clean_category['income_level'] = data_clean_category['income_level'].cat.codes
data_clean_category['marital_status'] = data_clean_category['marital_status'].cat.codes
data_clean_category['housing_status'] = data_clean_category['housing_status'].cat.codes
data_clean_category['employment'] = data_clean_category['employment'].cat.codes
data_clean_category['census_msa'] = data_clean_category['census_msa'].cat.codes

```


data_clean_category.head()										
	age_bracket	qualification	race	sex	income_level	marital_status	housing_status	employment	census_msa	
0	3	1	3	0	2	1	0	1	2	
1	1	0	3	1	2	1	1	0	0	
2	0	2	3	1	0	1	0	0	0	
3	4	0	3	0	2	1	1	1	1	
4	2	3	3	0	0	0	0	0	0	


```

data_clean_numeric.isna().sum()
h1n1_worry      0
h1n1_awareness  0
antiviral_medication  0
contact_avoidance  0
bought_face_mask  0
wash_hands_frequently  0
avoid_large_gatherings  0
reduced_outside_home_cont  0
avoid_touch_face  0
dr_recc_h1n1_vacc  0

```

```

reduced_outside_home_cont    0
avoid_touch_face              0
dr_recc_h1n1_vacc             0
dr_recc_seasonal_vacc         0
chronic_medic_condition       0
cont_child_unchr_6_mnth      0
is_health_worker              0
has_health_insur              0
is_h1n1_vacc_effective         0
is_h1n1_risky                  0
sick_from_h1n1_vacc           0
is_seas_vacc_effective         0
is_seas_risky                  0
sick_from_seas_vacc           0
no_of_adults                  0
no_of_children                0
h1n1_vaccine                   0
dtype: int64

```

```
[ ] data_clean_numeric = data_clean_numeric.apply(lambda x: x.fillna(x.value_counts().index[0]))
```

```
[ ] data_clean_numeric.isna().sum()
```

```

h1n1_worry                    0
h1n1_awareness                0
antiviral_medication          0
contact_avoidance              0
bought_face_mask              0
wash_hands_frequently         0
avoid_large_gatherings        0
reduced_outside_home_cont     0
avoid_touch_face              0
dr_recc_h1n1_vacc             0
dr_recc_seasonal_vacc         0
chronic_medic_condition       0
cont_child_unchr_6_mnth      0
is_health_worker              0
has_health_insur              0
is_h1n1_vacc_effective         0
is_h1n1_risky                  0
sick_from_h1n1_vacc           0
is_seas_vacc_effective         0
is_seas_risky                  0
sick_from_seas_vacc           0
no_of_adults                  0
no_of_children                0
h1n1_vaccine                   0
dtype: int64

```

```

sick_from_h1n1_vacc           0
is_seas_vacc_effective         0
is_seas_risky                  0
sick_from_seas_vacc           0
no_of_adults                  0
no_of_children                0
h1n1_vaccine                   0
dtype: int64

```

```
[ ] print(type(data_clean_category))
print(type(data_clean_numeric))

<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

Merging all data

Integrating category data with numeric data

```
[ ] covid_data = pd.concat([data_clean_category,data_clean_numeric],
                           join='outer',
                           axis=1)
covid_data.head()
```

	age_bracket	qualification	race	sex	income_level	marital_status	housing_status	employment	ce
0	3	1	3	0	2	1	0	1	
1	1	0	3	1	2	1	1	0	
2	0	2	3	1	0	1	0	0	
3	4	0	3	0	2	1	1	1	
4	2	3	3	0	0	0	0	0	

```
[ ] X = covid_data.iloc[:, :-1]
y = covid_data.iloc[:, -1]
```

```
[ ] X.shape,y.shape

((26707, 32), (26707,))
```

```
[ ] X = pd.get_dummies(X)
```

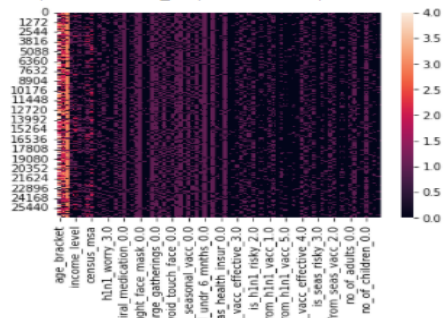
☞ (26707, 80)

```
[ ] X.head()
```

	age_bracket	qualification	race	sex	income_level	marital_status	housing_status	employment	census_msa	h1n1_worry_0.0	h1n1_wo
0	3	1	3	0	2	1	0	1	2	0	
1	1	0	3	1	2	1	1	0	0	0	
2	0	2	3	1	0	1	0	0	0	0	
3	4	0	3	0	2	1	1	1	1	0	
4	2	3	3	0	0	0	0	0	0	0	

```
[ ] sns.heatmap(X)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa3194b34d0>
```



```
covid = pd.concat([X,y],axis=1)
```

```
[ ] covid.head()
```

	age_bracket	qualification	race	sex	income_level	marital_status	housing_status	employment	census_msa	h1n1_worry_0_1
0	3	1	3	0	2	1	0	1	2	
1	1	0	3	1	2	1	1	0	0	
2	0	2	3	1	0	1	0	0	0	
3	4	0	3	0	2	1	1	1	1	
4	2	3	3	0	0	0	0	0	0	

5 rows $\times$ 81 columns

```
[ ] type(covid)
```

pandas.core.frame.DataFrame

```
[ ] X = covid.iloc[:, :-1].values #features
    y = covid.iloc[:, -1].values #target
```

```
[ ] X,y,X.shape,y.shape
```

```
(array([[3, 1, 3, ..., 0, 0, 0],
       [1, 0, 3, ..., 0, 0, 0],
       [0, 2, 3, ..., 0, 0, 0],
       ...,
       [3, 3, 3, ..., 0, 0, 0],
       [0, 3, 1, ..., 0, 0, 0],
       [4, 3, 3, ..., 0, 0, 0]], dtype=int16),
Length: 26707
Categories (2, int64): [0, 1],
(26707, 80),
(26707,))
```

```
Categories (2, int64): [0, 1],
(26707, 80),
(26707,))
```

Splitting the data

```
[ ] from sklearn.model_selection import train_test_split
```

```
Xtrain,Xtest,ytrain,ytest = train_test_split(X,y,test_size=0.2)
```

```
[ ] Xtrain.shape,Xtest.shape,ytrain.shape,ytest.shape
```

```
((21365, 80), (5342, 80), (21365,), (5342,))
```

```
[ ] from sklearn.linear_model import LogisticRegression
```

```
[ ] model = LogisticRegression()
```

```
[ ] model
```

```
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)
```

```
[ ] model.fit(Xtrain,ytrain)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, l1_ratio=None, max_iter=100,
```

```
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)
```

```
[ ] ypred = model.predict(Xtest)
```

```
[ ] ypred
```

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

```
[ ]
```

```
ytest
```

```
[0, 0, 0, 0, 0, ..., 0, 1, 0, 0, 0]
```

```
Length: 5342
```

```
Categories (2, int64): [0, 1]
```

```
[ ] from sklearn import model_selection
```

```
from sklearn.metrics import classification_report
```

```
from sklearn.metrics import confusion_matrix
```

```
from sklearn.metrics import accuracy_score
```

```
[ ] print(sklearn.metrics.classification_report(ytest,ypred))
```

	precision	recall	f1-score	support
0	0.85	0.95	0.90	4187
1	0.70	0.40	0.51	1155
accuracy			0.83	5342
macro avg	0.78	0.68	0.71	5342
weighted avg	0.82	0.83	0.82	5342

```
Vaccinators = pd.DataFrame(ypred)
Vaccinators.value_counts()
```

```
0    4677
1     665
dtype: int64
```

```
[ ] print('Default Logistic Regression Accuracy Training Score',model.score(Xtrain,ytrain))
print('Default Logistic Regression Accuracy Test Score',model.score(Xtest,ytest))
```

```
Default Logistic Regression Accuracy Training Score 0.8367423355956003
Default Logistic Regression Accuracy Test Score 0.8333957319356047
```

▼ Logistic Regression using MLE

```
[ ] covid.shape #Checking records and variables
```

```
(26707, 81)
```

```
[ ] import statsmodels.api as sm
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
import pandas.util.testing as tm
```

```
[ ] from statsmodels.formula.api import logit #We want to use Logistic Regression in StatsModels
```

```
[ ] covid_float = covid.astype('float') #statsmodels require dataset to have float values
```

```
[ ] X =covid_float.iloc[:, :-1]
```

```
y = covid_float.iloc[:, -1]
```

```
[ ] print(sklearn.metrics.confusion_matrix(ytest,ypred))
```

```
[[3987  200]
 [ 690  465]]
```

```
[ ] print(sklearn.metrics.accuracy_score(ytest,ypred))
```

```
0.8333957319356047
```

```
[ ] accuracy = (4041+444)/(4041+226+631+444)
accuracy
```

```
0.8395731935604642
```

```
[ ] model.coef_ #Weights or coefficient
```

```
array([[ 0.10990558,  0.01209893,  0.07441697,  0.16104742, -0.0042478 ,
        -0.18147597, -0.06354807,  0.03592878,  0.04865588,  0.11758761,
         0.02445273, -0.08740852, -0.22569478, -0.1297901 , -0.10546599,
         0.06419314, -0.15028699, -0.02077597, -0.08433423, -0.08672873,
        -0.18549907,  0.01443611, -0.0979321 , -0.07313085,  0.02520819,
        -0.19627114, -0.06942041, -0.10164255, -0.11237713, -0.05868583,
        -1.10741139,  0.93634844,  0.15877286, -0.32983581, -0.12340051,
        -0.04766245, -0.21299688,  0.04193392, -0.47710146,  0.30603851,
        -0.18272218,  0.01165922, -0.70012916, -0.64738562, -0.23111854,
         0.31477875,  1.09279162, -0.82512616, -0.3465111 ,  0.03822751,
         0.35805949,  0.60428731,  0.1634285 , -0.10859282, -0.44747236,
         0.07757304,  0.14400068, -0.22462897, -0.32531558,  0.29354678,
        -0.0113505 ,  0.09668531, -0.48893528, -0.11161037,  0.18534512,
         0.07541286,  0.16872472,  0.24474092,  0.12943588, -0.48268096,
         0.0350085 , -0.09756728, -0.02704075, -0.03205618, -0.06032668,
        -0.05163934,  0.00944811, -0.0340752 , -0.04946635, -0.09696952]])
```

```
[ ] model.intercept_
```

```
array([-0.20504855])
```

```

mle_model = logit("y~X",covid_float,.fit())
Warning: Maximum number of iterations has been exceeded.
Current function value: 0.382266
Iterations: 35
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:512: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
"Check mle_retvals", ConvergenceWarning)

[ ] print(mle_model.summary())

/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:1286: RuntimeWarning: invalid value encountered in sqrt
bse_ = np.sqrt(np.diag(self.cov_params()))
Logit Regression Results
=====
Dep. Variable:          y      No. Observations:      26707
Model:                Logit      Df Residuals:      26649
Method:                MLE      Df Model:          57
Date:                Fri, 05 Nov 2021      Pseudo R-squ.:      0.2609
Time:                11:08:13      Log-Likelihood:     -10209.
converged:              False      LL-Null:          -13813.
Covariance Type:      nonrobust      LIR p-value:       0.000
=====
                    coef    std err          z      P>|z|      [0.025      0.975]
-----
Intercept          -0.1726         nan         nan         nan         nan         nan
X[0]                0.1107         0.017         6.708         0.000         0.078         0.143
X[1]                0.0232         0.018         1.320         0.187        -0.011         0.058
X[2]                0.0664         0.022         3.002         0.003         0.023         0.110
X[3]                0.1639         0.038         4.326         0.000         0.090         0.238
X[4]                0.0060         0.028         0.216         0.829        -0.049         0.061
X[5]               -0.1435         0.050        -2.881         0.004        -0.241        -0.046
X[6]               -0.0601         0.049        -1.226         0.220        -0.156         0.036
X[7]                0.0312         0.034         0.932         0.351        -0.034         0.097
X[8]                0.0382         0.022         1.754         0.079        -0.004         0.081
X[9]                0.1049         4.13e+05         2.54e-07         1.000        -8.09e+05         8.09e+05
X[10]               0.0214         4.08e+05         5.23e-08         1.000        -8e+05         8e+05
X[11]               -0.0703         4.9e+05        -1.43e-07         1.000        -9.61e+05         9.61e+05
X[12]               -0.2285          3e+05        -7.62e-07         1.000        -5.88e+05         5.88e+05
X[13]               -0.1305         8.88e+04        -1.47e-06         1.000        -1.74e+05         1.74e+05
X[14]               -0.1018         2.44e+05        -4.18e-07         1.000        -4.77e+05         4.77e+05
X[15]                0.0597         nan         nan         nan         nan         nan
X[16]               -0.1891         7.12e+05        -2.66e-07         1.000        -1.39e+06         1.39e+06
X[17]                0.0164         8.07e+05         2.04e-08         1.000        -1.58e+06         1.58e+06
X[18]                0.0640         0.000         0.000         1.000         0.000         0.000

```

X[18]	-0.0640	nan	nan	nan	nan	nan
X[19]	-0.1086	nan	nan	nan	nan	nan
X[20]	-0.1906	6.76e+05	-2.82e-07	1.000	-1.32e+06	1.32e+06
X[21]	0.0179	6.27e+05	2.86e-08	1.000	-1.23e+06	1.23e+06
X[22]	-0.1020	nan	nan	nan	nan	nan
X[23]	-0.0706	nan	nan	nan	nan	nan
X[24]	0.0142	2.2e+06	6.46e-09	1.000	-4.31e+06	4.31e+06
X[25]	-0.1869	2.22e+06	-8.42e-08	1.000	-4.35e+06	4.35e+06
X[26]	-0.0645	nan	nan	nan	nan	nan
X[27]	-0.1082	nan	nan	nan	nan	nan
X[28]	-0.1113	nan	nan	nan	nan	nan
X[29]	-0.0613	nan	nan	nan	nan	nan
X[30]	-1.0919	2.97e+06	-3.68e-07	1.000	-5.82e+06	5.82e+06
X[31]	0.9193	2.95e+06	3.11e-07	1.000	-5.79e+06	5.79e+06
X[32]	0.1440	2.74e+06	5.25e-08	1.000	-5.38e+06	5.38e+06
X[33]	-0.3166	2.59e+06	-1.22e-07	1.000	-5.08e+06	5.08e+06
X[34]	-0.1267	nan	nan	nan	nan	nan
X[35]	-0.0460	nan	nan	nan	nan	nan
X[36]	-0.2137	1.13e+06	-1.9e-07	1.000	-2.21e+06	2.21e+06
X[37]	0.0411	1.1e+06	3.74e-08	1.000	-2.15e+06	2.15e+06
X[38]	-0.4933	1.26e+06	-3.91e-07	1.000	-2.47e+06	2.47e+06
X[39]	0.3207	1.25e+06	2.57e-07	1.000	-2.45e+06	2.45e+06
X[40]	-0.1873	1.91e+04	-9.82e-06	1.000	-3.74e+04	3.74e+04
X[41]	0.0146	6.92e+04	2.12e-07	1.000	-1.36e+05	1.36e+05
X[42]	-0.5981	1.98e+06	-3.02e-07	1.000	-3.89e+06	3.89e+06
X[43]	-0.7278	2.02e+06	-3.61e-07	1.000	-3.95e+06	3.95e+06
X[44]	-0.2283	2.01e+06	-1.13e-07	1.000	-3.95e+06	3.95e+06
X[45]	0.2811	1.97e+06	1.43e-07	1.000	-3.86e+06	3.86e+06
X[46]	1.1005	1.98e+06	5.57e-07	1.000	-3.87e+06	3.87e+06
X[47]	-0.8078	nan	nan	nan	nan	nan
X[48]	-0.3057	nan	nan	nan	nan	nan
X[49]	-0.0470	nan	nan	nan	nan	nan
X[50]	0.3598	nan	nan	nan	nan	nan
X[51]	0.6281	nan	nan	nan	nan	nan
X[52]	0.1993	2.4e+06	8.3e-08	1.000	-4.71e+06	4.71e+06
X[53]	-0.0720	2.42e+06	-2.98e-08	1.000	-4.74e+06	4.74e+06
X[54]	-0.5418	2.4e+06	-2.26e-07	1.000	-4.71e+06	4.71e+06
X[55]	0.1170	2.44e+06	4.79e-08	1.000	-4.78e+06	4.78e+06
X[56]	0.1249	2.41e+06	5.18e-08	1.000	-4.73e+06	4.73e+06
X[57]	-0.2389	nan	nan	nan	nan	nan
X[58]	-0.2964	nan	nan	nan	nan	nan

X[40]	-0.1873	1.91e+04	-9.82e-06	1.000	-3.74e+04	3.74e+04
X[41]	0.0146	6.92e+04	2.12e-07	1.000	-1.36e+05	1.36e+05
X[42]	-0.5981	1.98e+06	-3.02e-07	1.000	-3.89e+06	3.89e+06
X[43]	-0.7278	2.02e+06	-3.61e-07	1.000	-3.95e+06	3.95e+06
X[44]	-0.2283	2.01e+06	-1.13e-07	1.000	-3.95e+06	3.95e+06
X[45]	0.2811	1.97e+06	1.43e-07	1.000	-3.86e+06	3.86e+06
X[46]	1.1005	1.98e+06	5.57e-07	1.000	-3.87e+06	3.87e+06
X[47]	-0.8078	nan	nan	nan	nan	nan
X[48]	-0.3057	nan	nan	nan	nan	nan
X[49]	-0.0470	nan	nan	nan	nan	nan
X[50]	0.3598	nan	nan	nan	nan	nan
X[51]	0.6281	nan	nan	nan	nan	nan
X[52]	0.1993	2.4e+06	8.3e-08	1.000	-4.71e+06	4.71e+06
X[53]	-0.0720	2.42e+06	-2.98e-08	1.000	-4.74e+06	4.74e+06
X[54]	-0.5418	2.4e+06	-2.26e-07	1.000	-4.71e+06	4.71e+06
X[55]	0.1170	2.44e+06	4.79e-08	1.000	-4.78e+06	4.78e+06
X[56]	0.1249	2.41e+06	5.18e-08	1.000	-4.73e+06	4.73e+06
X[57]	-0.2389	nan	nan	nan	nan	nan
X[58]	-0.2964	nan	nan	nan	nan	nan
X[59]	0.3222	nan	nan	nan	nan	nan
X[60]	-0.0277	nan	nan	nan	nan	nan
X[61]	0.0681	nan	nan	nan	nan	nan
X[62]	-0.5354	1.81e+06	-2.96e-07	1.000	-3.55e+06	3.55e+06
X[63]	-0.1167	1.83e+06	-6.36e-08	1.000	-3.6e+06	3.6e+06
X[64]	0.2275	1.78e+06	1.28e-07	1.000	-3.49e+06	3.49e+06
X[65]	0.0903	1.83e+06	4.93e-08	1.000	-3.59e+06	3.59e+06
X[66]	0.1617	1.85e+06	8.76e-08	1.000	-3.62e+06	3.62e+06
X[67]	0.2443	nan	nan	nan	nan	nan
X[68]	0.1189	nan	nan	nan	nan	nan
X[69]	-0.4454	nan	nan	nan	nan	nan
X[70]	0.0643	nan	nan	nan	nan	nan
X[71]	-0.1548	nan	nan	nan	nan	nan
X[72]	-0.0355	2.66e+06	-1.34e-08	1.000	-5.21e+06	5.21e+06
X[73]	-0.0042	2.65e+06	-1.59e-09	1.000	-5.19e+06	5.19e+06
X[74]	-0.0512	2.63e+06	-1.94e-08	1.000	-5.16e+06	5.16e+06
X[75]	-0.0817	2.64e+06	-3.09e-08	1.000	-5.18e+06	5.18e+06
X[76]	0.0049	nan	nan	nan	nan	nan
X[77]	0.0177	nan	nan	nan	nan	nan
X[78]	-0.0832	nan	nan	nan	nan	nan
X[79]	-0.1120	nan	nan	nan	nan	nan

```
print(mle_model.summary2())
```

```
Results: Logit
=====
Model:                               Logit                               Pseudo R-squared: 0.261
Dependent Variable:                   y                               AIC: 20534.3752
Date: 2021-11-05 11:08                 BIC: 21009.5507
No. Observations: 26707                Log-Likelihood: -10209.
Df Model: 57                           LL-Null: -13813.
Df Residuals: 26649                    LLR p-value: 0.0000
Converged: 0.0000                       Scale: 1.0000
No. Iterations: 35.0000
=====
              Coef.      Std.Err.      z      P>|z|      [0.025      0.975]
-----
Intercept -0.1726          nan          nan          nan          nan          nan
X[0]       0.1107       0.0165      6.7083  0.0000      0.0783      0.1430
X[1]       0.0232       0.0175      1.3200  0.1868     -0.0112      0.0575
X[2]       0.0664       0.0221      3.0018  0.0027      0.0231      0.1098
X[3]       0.1639       0.0379      4.3262  0.0000      0.0897      0.2382
X[4]       0.0060       0.0280      0.2161  0.8289     -0.0488      0.0609
X[5]       -0.1435       0.0498     -2.8808  0.0040     -0.2411     -0.0459
X[6]       -0.0601       0.0490     -1.2265  0.2200     -0.1561      0.0359
X[7]       0.0312       0.0335      0.9320  0.3513     -0.0344      0.0969
X[8]       0.0382       0.0218      1.7537  0.0795     -0.0045      0.0808
X[9]       0.1049     412747.4899      0.0000  1.0000    -808970.1101    808970.3199
X[10]      0.0214     408260.8537      0.0000  1.0000    -800176.5482    800176.5909
X[11]      -0.0703     490270.1019     -0.0000  1.0000    -960911.8128    960911.6721
X[12]      -0.2285     299821.3758     -0.0000  1.0000    -587639.3268    587638.8698
X[13]      -0.1305     88799.6752     -0.0000  1.0000    -174044.2958    174044.0347
X[14]      -0.1018     243549.0292     -0.0000  1.0000    -477347.4276    477347.2240
X[15]       0.0597          nan          nan          nan          nan          nan
X[16]      -0.1891     711609.9123     -0.0000  1.0000   -1394729.9882   1394729.6101
X[17]       0.0164     807281.1096      0.0000  1.0000   -1582241.8838   1582241.9167
X[18]      -0.0640          nan          nan          nan          nan          nan
X[19]      -0.1086          nan          nan          nan          nan          nan
X[20]      -0.1906     675766.7503     -0.0000  1.0000   -1324478.6831   1324478.3020
X[21]       0.0179     627479.2287      0.0000  1.0000   -1229836.6713   1229836.7071
X[22]      -0.1020          nan          nan          nan          nan          nan
X[23]      -0.0706          nan          nan          nan          nan          nan
X[24]       0.0142     2201330.3288      0.0000  1.0000   -4314528.1482   4314528.1767
X[25]      -0.1869     2217976.3208     -0.0000  1.0000   -4347153.8941   4347153.5204
X[26]      -0.0645          nan          nan          nan          nan          nan
X[27]      -0.1082          nan          nan          nan          nan          nan
X[28]      -0.1082          nan          nan          nan          nan          nan
X[29]      -0.1113          nan          nan          nan          nan          nan
X[30]      -0.0613          nan          nan          nan          nan          nan
X[31]      -1.0919     2969300.0957     -0.0000  1.0000   -5819722.3388   5819720.1550
X[32]       0.9193     2954286.7428      0.0000  1.0000   -5790294.6966   5790296.5351
X[33]       0.1440     2742690.9954      0.0000  1.0000   -5375575.4277   5375575.7156
X[34]      -0.3166     2594081.4487     -0.0000  1.0000   -5084306.5290   5084305.8959
X[35]      -0.1267          nan          nan          nan          nan          nan
X[36]      -0.0460          nan          nan          nan          nan          nan
X[37]      -0.2137     1126519.0399     -0.0000  1.0000   -2207936.9599   2207936.5324
X[38]       0.0411     1098375.8003      0.0000  1.0000   -2152776.9689   2152777.0511
X[39]      -0.4933     1260430.3443     -0.0000  1.0000   -2470398.5732   2470397.5866
X[40]       0.3207     1249012.2046      0.0000  1.0000   -2448018.6166   2448019.2580
X[41]      -0.1873     19063.2379     -0.0000  1.0000    -37363.4469    37363.0724
X[42]       0.0146     69171.1843      0.0000  1.0000   -135573.0154   135573.0446
X[43]      -0.5981     1983477.8507     -0.0000  1.0000   -3887545.7497   3887544.5535
X[44]      -0.7278     2015954.3264     -0.0000  1.0000   -3951198.6021   3951197.1465
X[45]      -0.2283     2013222.9257     -0.0000  1.0000   -3945844.6556   3945844.1990
X[46]       0.2811     1970042.8018      0.0000  1.0000   -3861212.6585   3861213.2206
X[47]       1.1005     1975795.3104      0.0000  1.0000   -3872486.5487   3872488.7497
X[48]      -0.8078          nan          nan          nan          nan          nan
X[49]      -0.3057          nan          nan          nan          nan          nan
X[50]      -0.0470          nan          nan          nan          nan          nan
X[51]       0.3598          nan          nan          nan          nan          nan
X[52]       0.6281          nan          nan          nan          nan          nan
X[53]       0.1993     2401119.4824      0.0000  1.0000   -4706107.5088   4706107.9074
X[54]      -0.0720     2419583.3946     -0.0000  1.0000   -4742296.3831   4742296.2390
X[55]      -0.5418     2400576.5131     -0.0000  1.0000   -4705044.0496   4705042.9661
X[56]       0.1170     2439552.5336      0.0000  1.0000   -4781434.9872   4781435.2211
X[57]       0.1249     2412385.3411      0.0000  1.0000   -4728188.2604   4728188.5102
X[58]      -0.2389          nan          nan          nan          nan          nan
X[59]      -0.2964          nan          nan          nan          nan          nan
X[60]       0.3222          nan          nan          nan          nan          nan
X[61]      -0.0277          nan          nan          nan          nan          nan
X[62]       0.0681          nan          nan          nan          nan          nan
X[63]      -0.5354     1809014.0649     -0.0000  1.0000   -3545602.9501   3545601.8794
X[64]      -0.1167     1834475.8493     -0.0000  1.0000   -3595506.7117   3595506.4784
X[65]       0.2275     1781995.3536      0.0000  1.0000   -3492646.4863   3492646.9412
X[66]       0.0903     1830921.5930      0.0000  1.0000   -3588540.2917   3588540.4722
X[67]       0.1617     1845568.9900      0.0000  1.0000   -3617248.5896   3617248.9130
X[68]       0.2443          nan          nan          nan          nan          nan
X[69]       0.1189          nan          nan          nan          nan          nan
X[70]      -0.4454          nan          nan          nan          nan          nan
X[71]       0.0643          nan          nan          nan          nan          nan
X[72]      -0.1548          nan          nan          nan          nan          nan
X[73]       0.0255     2650116.7675      0.0000  1.0000   -5311777.1204   5311777.0602
```

```
X[72]      -0.1348      nan      nan      nan      nan      nan
X[72]      -0.0355 2659116.7675 -0.0000 1.0000 -5211773.1304 5211773.0593
X[73]      -0.0042 2645552.2217 -0.0000 1.0000 -5185187.0779 5185187.0695
X[74]      -0.0512 2632585.8216 -0.0000 1.0000 -5159773.4478 5159773.3454
X[75]      -0.0817 2641884.8695 -0.0000 1.0000 -5177999.2772 5177999.1138
X[76]       0.0049      nan      nan      nan      nan      nan
X[77]       0.0177      nan      nan      nan      nan      nan
X[78]      -0.0832      nan      nan      nan      nan      nan
X[79]      -0.1120      nan      nan      nan      nan      nan
=====
```

```
[ ] mle_model.pred_table() #Build confusion Matrix
```

```
array([[19942., 1091.],
       [ 3276., 2398.]])
```

```
[ ] mle_model.aic
```

```
20534.375217681893
```

```
[ ] accuracy = (19942+2398)/(19942+1091+3276+2398)
accuracy
```

```
0.836484816714719
```

▼ Logistic Regression using SGD Model

```
[ ] from sklearn.linear_model import SGDClassifier
sgd_classifier = SGDClassifier()
sgd_classifier
```

```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
```

```
3.18330855e-01, 3.50813595e-01, 2.92344663e-01,
-4.87241104e-01, 1.13039936e+00, 4.74248008e-01,
1.68910250e-01, 2.14386086e-01, 4.28772172e-01,
1.94896442e-01, 4.48261816e-01, -7.38906290e-16,
6.43158250e-01, 3.05337759e-01, 3.37828499e-01,
-2.27379182e-01, -2.53365374e-01, -3.89792884e-02,
3.11834307e-01, 8.51047796e-01, -4.54758364e-01,
-1.55917153e-01, 7.79585767e-02, 4.87241104e-01,
6.88634094e-01, 2.07889538e-01, 7.14620287e-02,
3.24827403e-02, 1.29930961e-02, 3.18330855e-01,
1.94896442e-02, -4.54758364e-02, 3.44317047e-01,
1.10441317e-01, 2.14386086e-01, -1.23434413e-01,
-1.94896442e-02, 1.88399894e-01, 1.88399894e-01,
4.09282528e-01, 3.89792884e-01, 1.49420605e-01,
-1.55917153e-01, 1.68910250e-01, 9.09516720e-02,
1.81903346e-01, 1.68910250e-01, 1.81903346e-01,
1.10441317e-01, 2.14386086e-01, 1.81903346e-01,
7.79585767e-02, 1.68910250e-01]])
```

```
[ ] sgd_clf.intercept_
```

```
array([-6.47684832])
```

```
[ ] sgd_clf.decision_function(Xtest)
```

```
array([-1.5264787, -2.11766457, -0.60396887, ..., -2.47497471,
       -1.72137514, -1.48100286])
```

```
[ ] sgd_clf = SGDClassifier(alpha=0.0001, average=False, class_weight=None,
                           early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
                           l1_ratio=0.15, learning_rate='optimal', loss='log',
                           max_iter=5, n_iter_no_change=5, n_jobs=None, penalty='l2',
                           power_t=0.5, random_state=None, shuffle=True, tol=0.001,
                           validation_fraction=0.1, verbose=0, warm_start=False).fit(Xtrain,ytrain)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_stochastic_gradient.py:557: ConvergenceWarning: Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.
ConvergenceWarning)
```

```
sgd_clf = SGDClassifier(loss='hinge',
                        penalty='l2',
                        max_iter=1000)
```

```
[ ] sgd_clf
```

```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

```
[ ] sgd_clf.fit(Xtrain,ytrain)
```

```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

```
[ ] print('Training Accuracy Score',sgd_clf.score(Xtrain,ytrain))
print('Validation Accuracy Score',sgd_clf.score(Xtest,ytest))
```

```
Training Accuracy Score 0.8336531710741868
Validation Accuracy Score 0.8304005990265818
```

```
[ ] sgd_clf.predict(Xtest)
```

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

```
[ ] sgd_clf.coef_
```

```
array([[ 1.94896442e-02,  6.49654806e-03,  9.09516728e-02,
         1.29930961e-01,  9.74482209e-02, -7.79585767e-02,
        -1.55917153e-01,  4.54758364e-02,  1.23434413e-01])
```

```
sgd_clf.predict_proba(Xtest)
```

```
array([[0.9789611, 0.0210389],
       [0.98605094, 0.01394906],
       [0.54408416, 0.45591584],
       ...,
       [0.99748953, 0.00251047],
       [0.99411664, 0.00588336],
       [0.99328953, 0.00671047]])
```

```
[ ] print(sgd_clf.predict_proba(Xtest))
```

```
[[0.9789611 0.0210389]
 [0.98605094 0.01394906]
 [0.54408416 0.45591584]
 ...
 [0.99748953 0.00251047]
 [0.99411664 0.00588336]
 [0.99328953 0.00671047]]
```

```
[ ] sgd_ypred = sgd_clf.predict(Xtest)
```

```
[ ] sklearn.metrics.accuracy_score(ytest,sgd_ypred)
```

```
0.8283414451516286
```

Compare Various Models

```
[ ] #Step 5: Build models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

```

from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

# Spot Check Algorithms
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
models.append(('DecisionTree', DecisionTreeClassifier()))
models.append(('Random Forest', RandomForestClassifier()))

```

```

[ ] # evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=None)
    cv_results = model_selection.cross_val_score(model, Xtrain, ytrain, cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)

```

```

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg= LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg= LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg= LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg= LOGISTIC_SOLVER_CONVERGENCE_MSG)
LR: 0.835291 (0.005347)
LDA: 0.833746 (0.005621)
KNN: 0.810391 (0.004236)
CART: 0.751744 (0.009424)
NB: 0.699742 (0.009179)
SVM: 0.835385 (0.003029)
DecisionTree: 0.751557 (0.008242)
Random Forest: 0.831969 (0.006930)

```
[ ] #Now we can make prediction on most accurate model
# Make predictions on validation dataset
rf = RandomForestClassifier()
rf.fit(Xtrain, ytrain)
predictions = rf.predict(Xtest)
print(accuracy_score(ytest, predictions))
```

▶ #Now we can make prediction on most accurate model
Make predictions on validation dataset
rf = RandomForestClassifier()
rf.fit(Xtrain, ytrain)
predictions = rf.predict(Xtest)
print(accuracy_score(ytest, predictions))
print(confusion_matrix(ytest, predictions))
print(classification_report(ytest, predictions))

```
0.827218270310745
[[3967  229]
 [ 694  452]]
      precision    recall  f1-score   support

     0       0.85       0.95       0.90       4196
     1       0.66       0.39       0.49       1146

 accuracy                   0.83       5342
 macro avg       0.76       0.67       0.70       5342
 weighted avg    0.81       0.83       0.81       5342
```

PREDICTIONS USING LOGISTIC REGRESSION AND IN RANDOM FOREST GAVE THE MOST ACCURACY RATE

A. SOURCE CODE

IMPORTING ALL THE NECESSARY LIBRARIES FOR VACCINE PREDICTION:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
data = pd.read_csv('/content/h1n1_vaccine_prediction.csv')
data.dtypes
data = data.drop(columns=['unique_id'])
data.columns
data_category_column = data.columns[data.dtypes=='object'] #which of the columns are category type
data_numeric_column = data.columns[data.dtypes!='object'] #which of the columns are of numeric type
print(data_category_column) #Names of category columns
print(data_numeric_column) #Names of numeric columns
Create data subsets
data_category = data[data_category_column] #Category data
data_numeric = data[data_numeric_column] #Numeric data
data_category.shape
data_numeric.shape
data_category.head()
data_numeric.head()

Check for missing data
data_category.isna().sum().sort_values(ascending=False)
data_numeric.isna().sum().sort_values(ascending=False)

Categorical imputation using lambda x:x.fillna(x.value_counts().index[0])
data_clean_category = data_category.apply(lambda x: x.fillna(x.value_counts().index[0]))
data_clean_category.isna().sum().sort_values(ascending=False)
data_clean_category.dtypes

use a loop to convert all object columns to category
for column in data_clean_category.columns:
    data_clean_category[column] = data_clean_category[column].astype('category')
data_clean_category.dtypes
data_clean_category.head()

Numeric columns missing value imputation
data_numeric.head()
```

```

data_numeric.dtypes
data_clean_numeric = data_numeric.apply(lambda x: x.fillna(x.value_counts().index[0]))
data_clean_numeric.head()
data_clean_numeric.isna().sum()
data_clean_numeric.dtypes
data_clean_numeric = data_clean_numeric.astype('category')
data_clean_numeric.head()
data_clean_numeric['h1n1_worry'].value_counts()
category_count = data_clean_category.columns.value_counts()
sns.barplot(category_count.index,category_count.values,alpha=0.9)
plt.show()
numeric_count = data_clean_numeric.columns.value_counts()
sns.barplot(numeric_count.index,numeric_count.values,alpha=0.9)
plt.show()
data_clean_category.columns
data_clean_numeric.columns
data_clean_category.dtypes
data_clean_numeric.dtypes
data_clean_category
data_clean_category['age_bracket'] = data_clean_category['age_bracket'].cat.codes

data_clean_category['qualification'] = data_clean_category['qualification'].cat.codes

data_clean_category['race'] = data_clean_category['race'].cat.codes

data_clean_category['sex'] = data_clean_category['sex'].cat.codes

data_clean_category['income_level'] = data_clean_category['income_level'].cat.codes

data_clean_category['marital_status'] = data_clean_category['marital_status'].cat.codes

data_clean_category['housing_status'] = data_clean_category['housing_status'].cat.codes

data_clean_category['employment'] = data_clean_category['employment'].cat.codes

data_clean_category['census_msa'] = data_clean_category['census_msa'].cat.codes
data_clean_category.head()
data_clean_numeric.isna().sum()
data_clean_numeric = data_clean_numeric.apply(lambda x: x.fillna(x.value_counts().index[0]))
data_clean_numeric.isna().sum()
print(type(data_clean_category))
print(type(data_clean_numeric))
Merging all data
Integrating category data with numeric data

```

```

covid_data = pd.concat([data_clean_category,data_clean_numeric],
join='outer',
                        axis=1)
covid_data.head()
X = covid_data.iloc[:, :-1]
y = covid_data.iloc[:, -1]
X.shape,y.shape
X = pd.get_dummies(X)
X.shape
X.head()
sns.heatmap(X)
covid = pd.concat([X,y],axis=1)
covid.head()
type(covid)
X = covid.iloc[:, :-1].values #features
y = covid.iloc[:, -1].values #target
X,y,X.shape,y.shape

```

Splitting the data

```

from sklearn.model_selection import train_test_split
Xtrain,Xtest,ytrain,ytest = train_test_split(X,y,test_size=0.2)
Xtrain.shape,Xtest.shape,ytrain.shape,ytest.shape
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model
model.fit(Xtrain,ytrain)
ypred = model.predict(Xtest)
ypred
ytest
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
print(sklearn.metrics.classification_report(ytest,ypred))
print(sklearn.metrics.confusion_matrix(ytest,ypred))
print(sklearn.metrics.accuracy_score(ytest,ypred))
accuracy = (4041+444)/(4041+226+631+444)
accuracy
model.coef_ #Weights or coefficient
model.intercept_
Vaccinators = pd.DataFrame(ypred)
Vaccinators.value_counts()
print('Default Logistic Regression Accuracy Training Score',model.score(Xtrain,ytrain))
print('Default Logistic Regression Accuracy Test Score',model.score(Xtest,ytest))

```

Logistic Regression using MLE

covid.shape #Checking records and variables

import statsmodels.api as sm

from statsmodels.formula.api import logit #We want to use Logistic Regression in StatsModels

covid_float = covid.astype('float') #statsmodels require dataset to have float values

X =covid_float.iloc[:, :-1]

y = covid_float.iloc[:, -1]

mle_model = logit("y~X",covid_float.).fit()

print(mle_model.summary())

print(mle_model.summary2())

mle_model.pred_table() #Build confusion Matrix

mle_model.aic

accuracy = (19942+2398)/(19942+1091+3276+2398)

accuracy

Logistic Regression using SGD Model

from sklearn.linear_model import SGDClassifier

sgd_classifier = SGDClassifier()

sgd_classifier

sgd_clf = SGDClassifier(loss='hinge',

penalty='l2',

max_iter=1000)

sgd_clf

sgd_clf.fit(Xtrain,ytrain)

print('Training Accuracy Score',sgd_clf.score(Xtrain,ytrain))

print('Validation Accuracy Score',sgd_clf.score(Xtest,ytest))

sgd_clf.predict(Xtest)

sgd_clf.coef_

sgd_clf.intercept_

sgd_clf.decision_function(Xtest)

sgd_clf = SGDClassifier(alpha=0.0001, average=False, class_weight=None,

early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,

l1_ratio=0.15, learning_rate='optimal', loss='log',

max_iter=5, n_iter_no_change=5, n_jobs=None, penalty='l2',

power_t=0.5, random_state=None, shuffle=True, tol=0.001,

validation_fraction=0.1, verbose=0, warm_start=False).fit(Xtrain,ytrain)

sgd_clf.predict_proba(Xtest)

print(sgd_clf.predict_proba(Xtest))

sgd_ypred = sgd_clf.predict(Xtest)

sklearn.metrics.accuracy_score(ytest,sgd_ypred)

Compare Various Models

#Step 5: Build models

from sklearn.linear_model import LogisticRegression

```

from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

# Spot Check Algorithms
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
models.append(('DecisionTree', DecisionTreeClassifier()))
models.append(('Random Forest', RandomForestClassifier()))
# evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=None)
    cv_results = model_selection.cross_val_score(model, Xtrain, ytrain, cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
#Now we can make prediction on most accurate model
# Make predictions on validation dataset
rf = RandomForestClassifier()
rf.fit(Xtrain, ytrain)
predictions = rf.predict(Xtest)
print(accuracy_score(ytest, predictions))

print(confusion_matrix(ytest, predictions))

print(classification_report(ytest, predictions))

```