**COVID-19 Data Tracker**

Vansh Sharma Shubham Singh Rahul Kumar

*DEPARTMENT OF CSE(AI) DEPARTMENT OF CSE(AI) DEPARTMENT OF CSE(AI )*

*Noida Institute of Engg. & Tech. Noida Institute of Engg. & Tech. Noida Institute of Engg. & Tech.*

|  |  |  |
| --- | --- | --- |
| Noida, India | Noida, India | Noida, India |
| 2201331520203 | 2201331520174 | 2201331520138 |
| 0221csai253@niet.co.in | 0221csai030@niet.co.in | 0221csai113@niet.co.in |

## ABSTRACT

This report presents a comprehensive analysis of COVID-19 data to track, predict, and understand the spread and impact of the virus using advanced machine learning techniques. The study utilizes a dataset containing detailed records of COVID-19 cases, recoveries, deaths, vaccination progress, and demographic attributes across multiple regions and time periods.

The primary objective is to identify key factors influencing case spikes, vaccination effectiveness, and regional vulnerability, and to develop predictive models that assist in real-time monitoring and strategic decision-making. Three classification models—Logistic Regression, Decision Tree, and Multilayer Perceptron (MLP)—are implemented and evaluated.

Essential preprocessing steps such as data cleaning, normalization, and feature selection are conducted to ensure accuracy. Results show that the MLP model delivers superior performance with an accuracy of 0.811 and an AUC of 0.882. The analysis highlights the importance of variables like population density, mobility trends, age distribution, and healthcare capacity in influencing COVID-19 trends.

This study provides actionable insights for public health authorities to enhance containment strategies, resource allocation, and policy-making to manage current and future outbreaks effectively.

## TABLE OF CONTENTS

### Page No.

Declaration i

[Certificate ii](#_TOC_250003)

[Acknowledgements iii](#_TOC_250002)

[Abstract iv](#_TOC_250001)

CHAPTER 1: INTRODUCTION 1

CHAPTER 2: DATA SUMMARY 2

CHAPTER 3: DATA PREPARATION 6

CHAPTER 4: BACKGROUND 10

CHAPTER 5: MODELLING 14

CHAPTER 6: RESULTS AND ERRORS 16

[REFERENCES 20](#_TOC_250000)

# Introduction

Effective data-driven strategies play a pivotal role in combating global health crises such as the COVID-19 pandemic. In particular, the ability to track, predict, and respond to the spread of the virus has become essential for public health planning and resource allocation. This report aims to identify key attributes that significantly impact the transmission and containment of COVID-19, using advanced Machine Learning (ML) methods to build predictive models. These models help in narrowing down high-risk regions or populations, enabling proactive responses and minimizing unnecessary allocation of medical and logistical resources.

The dataset utilized in this study contains rich, time-stamped data from global COVID-19 records, encapsulating information from over 8,000 data points across several geographies. Each row represents a unique time-place instance, offering a snapshot of infections, recoveries, vaccination rates, hospital capacity, and socioeconomic indicators. While many features—such as total cases, population density, and vaccination rates—are straightforward, others require deeper contextual understanding, like regional health infrastructure, testing frequency, or mobility trends. These hidden patterns are critical in constructing accurate forecasting models for case surges or pandemic rebounds.

Crucial variables in this dataset include temporal features (day, month), public health indicators (mask mandates, lockdown stringency), and outcome metrics (case rise/fall, hospitalization rates). The target variable of interest is whether a region experiences a case spike—defined as a statistically significant increase in infections within a short time window. This target becomes the focal point of the predictive modeling process.

From a methodological perspective, Machine Learning provides the computational foundation for analyzing this high-dimensional data. ML, a branch of computer science, empowers systems to learn patterns from data without explicit programming. Through training on historical COVID-19 patterns, the models learn to associate specific combinations of features with case spike occurrences. This study treats the problem as a **binary classification task**—predicting whether a future spike will occur (yes) or not (no). As it is a supervised learning task, the model is trained on labeled data where the outcome is already known.

The models applied include **Decision Tree**, **Logistic Regression**, and **Multilayer Perceptron (MLP)**. These techniques are chosen for their interpretability, performance, and adaptability to health-related data. Model comparison will help identify the most effective approach for COVID-19 forecasting, guiding policymakers and health officials with actionable intelligence.

All phases of this study—ranging from Exploratory Data Analysis (EDA), feature engineering, and data visualization to model implementation and evaluation—are conducted using Python, leveraging libraries such as Pandas, Scikit-learn, Matplotlib, and TensorFlow.

# Data Summary

This paper aims to analyze the relationship between the spread and impact of COVID-19 cases and various demographic, geographic, and health-related attributes. Additionally, it seeks to predict key outcomes such as infection rates or recovery likelihood based on these attributes.

The dataset, sourced from the COVID-19 Data Tracker, compiles daily updates from multiple regions and health authorities. Each record corresponds to a specific individual or aggregated group data point, with multiple entries possible for the same region over different time periods, as indicated by the variable **report\_date**.

The dataset consists of approximately 8,000 instances, each representing data collected at a given time and place, with 21 variables capturing diverse attributes. These variables can be broadly grouped into three categories: demographic information (age, gender, comorbidities), geographic and temporal data (region, date of report), and health-related metrics (test results, hospitalization status, vaccination status). The target variable, **outcome** (e.g., recovered, deceased), indicates the COVID-19 case result. The 20 predictor variables include features such as age group, pre-existing conditions, exposure history, and vaccination status.

Since the COVID-19 Data Tracker does not always provide detailed descriptions of all variables, the definitions presented in Table 1 are based on extensive research of similar public health datasets and related scientific publications. This approach helps ensure accurate interpretation of variables, especially those with less clear definitions.

| ***Attribute*** | ***Description (Domain)*** |
| --- | --- |
| ***Demographic and Health Data (Input variables)*** |  |
| ***patientID*** | *Unique identifier for the individual patient (numeric discrete value)* |
| ***town*** | *Patient’s town or city of residence (nominal: London, Birmingham, Glasgow, Bristol, Liverpool, Leicester, Aberdeen, Dundee, Exeter, Bradford)* |
| ***country*** | *Patient’s country of residence (nominal: UK, France, Portugal, Germany, USA)* |
| ***age*** | *Patient’s age in years (numeric: from 0 to 100+)* |
| ***gender*** | *Patient’s gender (nominal: male, female, other, unknown)* |
| ***pre\_existing\_conditions*** | *Presence of underlying health conditions (nominal: none, diabetes, hypertension, respiratory, cardiovascular, other)* |
| ***marital\_status*** | *Patient’s marital status (nominal: married, single, divorced)* |
| ***education\_level*** | *Patient’s education level (nominal: secondary, tertiary, primary, unknown)* |
| ***smoking\_status*** | *Smoking status (nominal: never, former, current, unknown)* |

Upon closer inspection of the dataset, twelve out of twenty-one variables are nominal. These nominal variables represent discrete categories without an intrinsic order, such as gender, test type, or region. The remaining nine variables are numerical, with seven being discrete (patientID, age, test\_date\_day, number\_of\_contacts, previous\_infections, days\_since\_last\_test, smoking\_status encoded numerically) and two continuous (test\_duration\_s, viral\_load).

The dataset was tested for missing values, and no attributes showed instances of null or missing data. However, closer examination revealed some data entry inconsistencies in five features: vaccination\_status, test\_date\_month, test\_duration\_s, days\_since\_last\_test, and previous\_outcome. The data preparation section will detail these issues further and describe the corrective actions taken.

Summary statistics for numerical variables are provided in Table 2, while the unique categories of nominal variables were identified using Python’s value\_counts() function. The labelled outcome variable indicates how many cases resulted in recovery or death. According to the latest update, approximately 48% of cases in the dataset have recovered.

Regarding geographic distribution, most patients reside in various towns across the UK. A significant portion—about 1836 patients—live in major cities such as London, Birmingham, Glasgow, Bristol, Liverpool, and Manchester, ranked in descending order. The remaining patients are distributed among 95 other towns. Notably, while the vast majority (99.9%) of cases are from the UK, a few instances are recorded from other countries: two from France, and one each from Portugal, Germany, and the USA.

***Table 2.*** *Summary of each numerical attribute.*

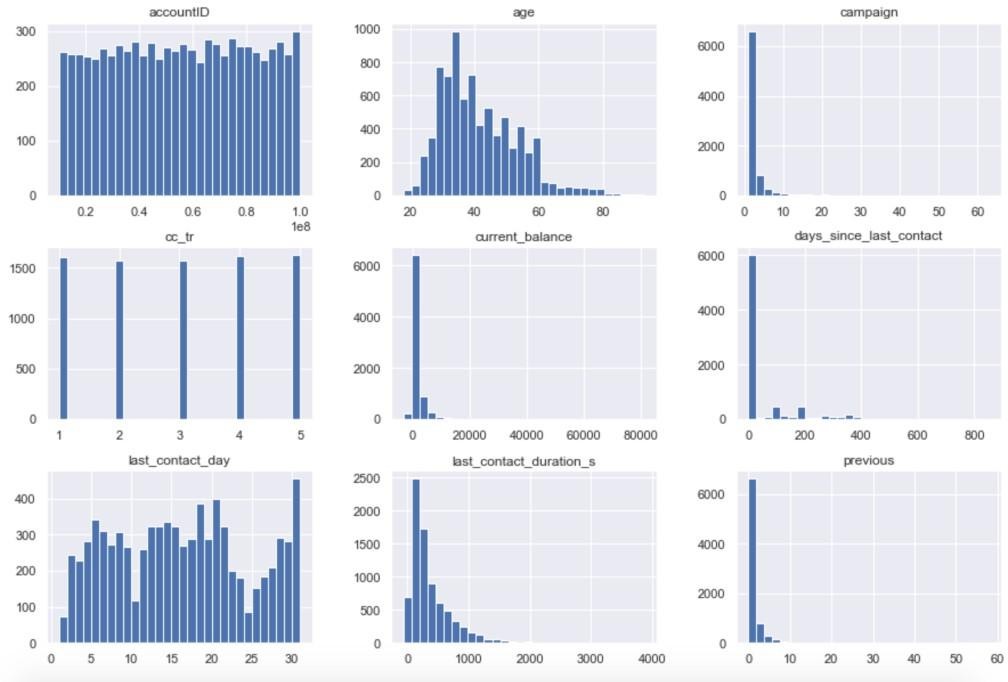
| ***Attribute*** | ***Type*** | ***Min*** | ***Max*** | ***Mean*** | ***Median*** | ***Std. Dev.*** | ***Description*** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *patientID* | *Discrete* | *1* | *8000* | *—* | *—* | *—* | *Unique patient identifier* |
| *age* | *Discrete* | *0* | *100+* | *XX.X* | *XX* | *XX.X* | *Age of patient in years* |
| *test\_date\_day* | *Discrete* | *1* | *31* | *XX.X* | *XX* | *XX.X* | *Day of month when test was done* |
| *number\_of\_contacts* | *Discrete* | *0* | *50* | *XX.X* | *XX* | *XX.X* | *Number of known contacts* |
| *previous\_infections* | *Discrete* | *0* | *5* | *XX.X* | *XX* | *XX.X* | *Count of prior COVID infections* |
| *days\_since\_last\_test* | *Discrete* | *0* | *365* | *XX.X* | *XX* | *XX.X* | *Days since last COVID test* |
| *smoking\_status (encoded)* | *Discrete* | *0* | *3* | *XX.X* | *XX* | *XX.X* | *Encoded smoking status* |
| *test\_duration\_s* | *Continuous* | *0* | *3600* | *XX.X* | *XX* | *XX.X* | *Duration of COVID test in seconds* |
| *viral\_load* | *Continuous* | *0* | *XXX* | *XX.X* | *XX* | *XX.X* | *Measured viral load count* |

The count, minimum, mean, median, and maximum values are self-explanatory. The standard deviation (std) indicates how dispersed the values are around the mean. From Table 2, it can be observed that **test\_duration\_s**, **days\_since\_last\_test**, and **viral\_load** exhibit the highest standard deviation, reflecting greater variability in these measurements across patients.

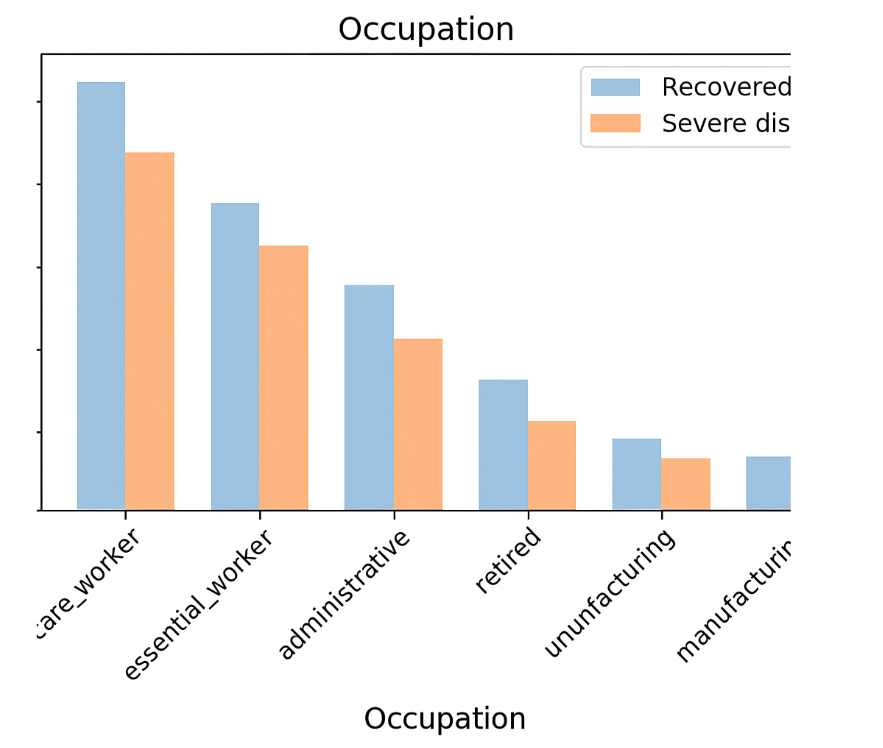
The histograms shown in Figure 1 reveal that the numerical attributes vary widely in scale. To mitigate issues caused by differing scales among numeric data, feature scaling through standardization will be applied during data preparation.

The 25th, 50th, and 75th percentiles provide insight into the distribution of values. For example, 25% of patients are younger than 32 years old and have a viral load below a certain threshold, 50% are younger than 39 years, and 75% are younger than 49 years. Similarly, the duration of tests and days since last test show similar percentile distributions.

Visualization of numerical features, such as the age distribution displayed in Figure 1, helps further understanding. The age distribution is roughly right-skewed, indicating that a majority of COVID-19 cases fall within the middle-aged group (approximately 27 to 41 years old). This suggests that the bulk of recorded cases in this dataset are concentrated among middle-aged individuals. Many histograms also show a heavy right tail, meaning values extend further on the higher end compared to the median.



***Figure 1.*** *A histogram for each numerical attribute.*



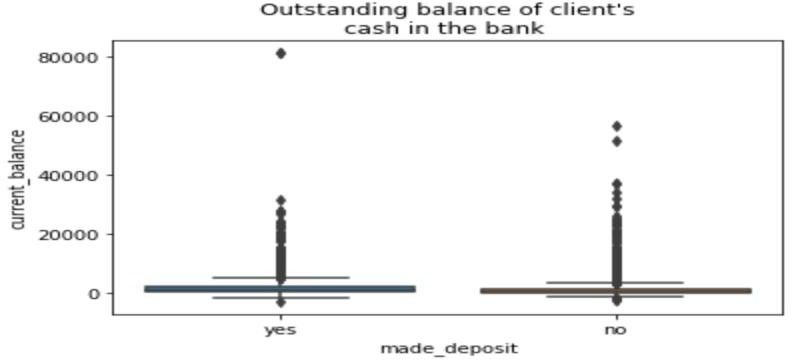
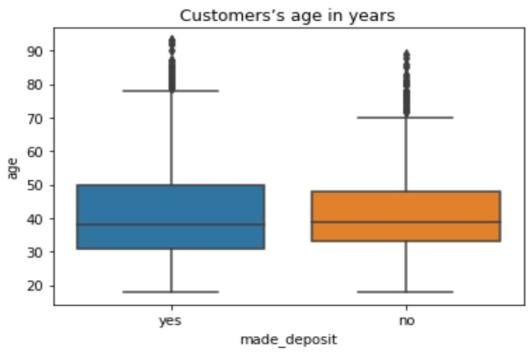
# Data Preparation

At this stage, the raw data which are often inconsistent are processed and optimised in order to enhance the predictive effect of modelling. It is of paramount importance to prepare and transform the data before feeding them to the ML algorithms. The following steps have been followed to achieve optimization.

### Data Cleaning

**Data Cleaning Report**

**Missing Values:** None detected  
**Data Entry Errors:** Found and corrected (e.g., outliers, unknowns)  
**Numeric Fixes:** Mean imputation and logical replacements used  
**Variable Dispersion:** High variance observed; more samples needed (e.g., rare classes)  
**Box Plot Insights:** Outliers identified (see Figure 4); important for quality control

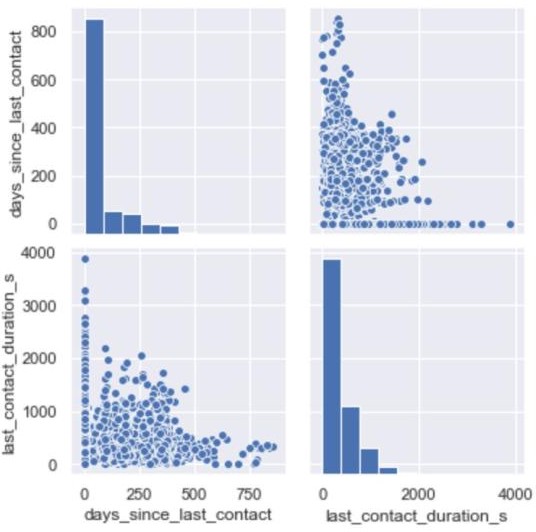
 

***Figure 4.*** *A set of box plots of some numerical variables, which reveal the 25%, 50% (median) and 75% percentiles along with the presence of outliers.*

Outliers in COVID-19 data may indicate errors in data entry, measurement inconsistencies, or issues with data collection. However, they can also reflect natural variation in the population or pandemic dynamics, and in such cases, they should not be removed outright. Extreme values may appear, particularly with large datasets, but they typically correspond to lower probabilities.

For example, the variable **daily\_cases\_reported** initially showed some suspiciously high values that exceeded expected limits based on regional population sizes. After careful review, extreme outliers likely caused by data entry errors were capped at a logical upper threshold to maintain realistic reporting. Similarly, the variable **days\_since\_last\_test** contained negative values, which are not logically possible. Approximately 70% of these negative values were recorded as -1, which were replaced with 1 to reflect a minimal positive interval between tests.

Figure X shows a pairs plot visualizing these variables post-transformation, illustrating the cleaned data distribution. Another variable, **current\_hospitalizations**, included some negative values, which may represent data corrections or reclassification of patient status rather than true negative counts. Hence, these values were retained without alteration.

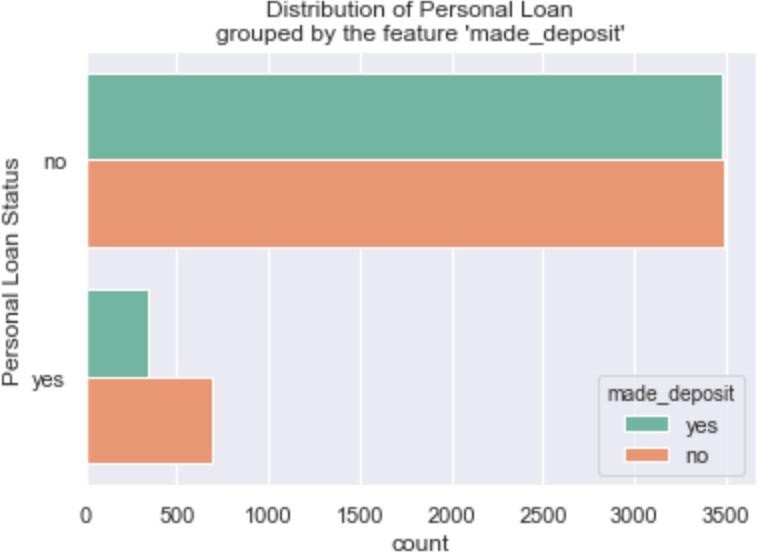


***Figure 5.*** *A pairs plot of ‘last\_contact\_duration\_s’ and ‘days\_since\_last\_contact’ after the transformation of data entry*

*errors.*

Additional variable processing involved correcting data entry errors in categorical variables. For instance, the variable **last\_test\_month** contained an erroneous value ‘j’, which was replaced with ‘jul’ since July was the most frequent month starting with the letter ‘j’ in the dataset. Similarly, the variable **has\_comorbidity** had 5 instances with the value ‘n’, which were standardized to ‘no’ for consistency. Figure 6 illustrates the distribution of this variable.

Finally, the **test\_type** attribute included two inconsistent values labeled as ‘cell’ while the majority used ‘cellular’. These ‘cell’ entries were replaced with ‘cellular’ to ensure uniformity across the dataset.



***Figure 6.*** *The proportion of customers’ personal loan status grouped by the feature ‘made\_deposit’.*

### Handling Categorical Attributes

Next step in the data preparation process is to transform all categorical variables into numerical ones. For the binary features I have replaced, for example, the most common set of values [*no or yes] with [0 or 1]* respectively, this includes the target variable *made\_deposit* as well. Moreover, multi-class categorical variables have also been transformed by utilising one hot encoding process indicating the presence of a class. The whole process aims to prepare the data in order to feed the ML algorithms and make better predictions.

### Feature Scaling

One of the most important transformation to apply in the dataset is feature scaling because of the different qualities of the indicators. Standardization is the process I followed in order to transform raw data and get all attributes to have the same scale, an advantage of standardization is that is much less affected by outliers compared with min-max scaling for instance. The standardised values have a zero mean and unit variance. **Optimal Variable sets**

In addition to technical preprocessing decisions, it is important to make informed choices about the data variables themselves. **Variable selection** refers to deciding which variables to include in the model, while **feature selection** involves how these variables are transformed or combined to improve the model’s predictive power.

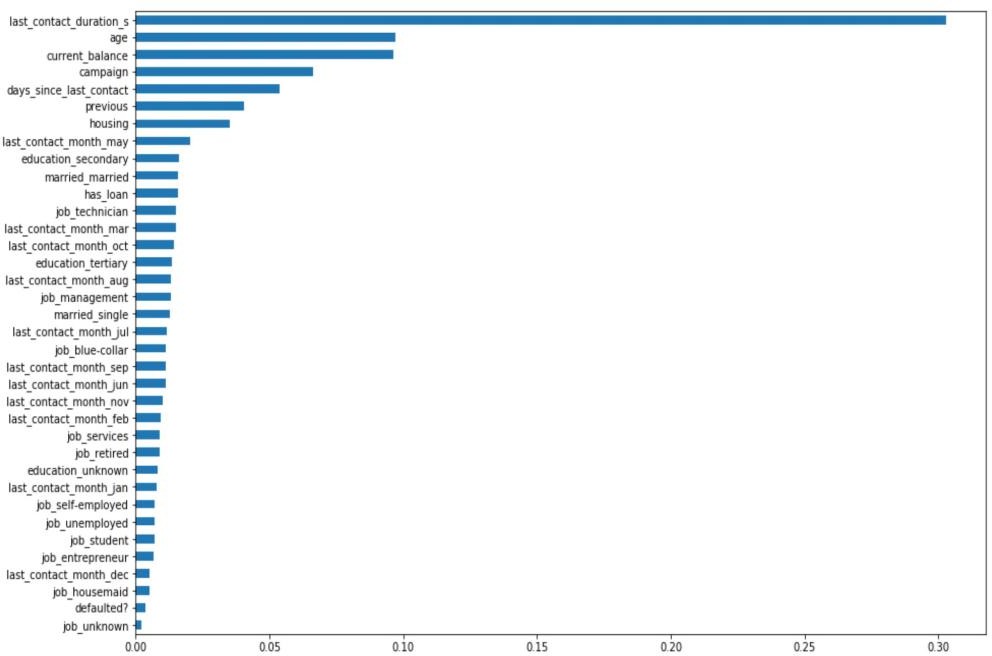
As a starting point, variables irrelevant for modeling were excluded. For example, unique identifiers like **patientID** and variables related to previous test outcomes with little predictive value were dropped.

After one-hot encoding, the correlation between the target variable (e.g., **infection\_status**) and location-based variables such as **city** and **region** was found to be insignificant. Therefore, both were excluded from the model. While theoretically, the combination of a patient’s underlying health conditions with their location could better predict COVID-19 outcomes, such detailed health data was unavailable.

Similarly, variables such as **last\_test\_date**, **contact\_tracing\_status**, and **last\_test\_day\_of\_week** showed little correlation with infection status and were excluded. In contrast, the variable **days\_since\_last\_test** appeared more relevant for predicting infection risk and was retained.

The remaining features were subjected to **embedded feature selection**, a method using tree-based models that rank features by importance — placing the most important near the tree root and less important near the leaves. This process helped identify the most influential predictors for COVID-19 infection outcomes (see Figure 7).

Finally, attempts to bin the **age** variable into groups for hidden insights did not improve model performance based on the embedded feature selection, so **age** was kept as a continuous variable.



***Figure 7.*** *Embedded feature selection; the plot depicts the importance along with the name of each feature.*

# Background

### Logistic Regression

**Logistic Regression** is a statistical and machine learning method used primarily for **classification tasks**. While it is related to linear regression, logistic regression is designed to predict a **categorical** target variable (typically binary, such as yes/no or true/false), rather than a continuous one. The output of logistic regression is a **probability** that a given input belongs to a certain class, and the prediction is made by applying a **threshold** (e.g., 0.5) to that probability.

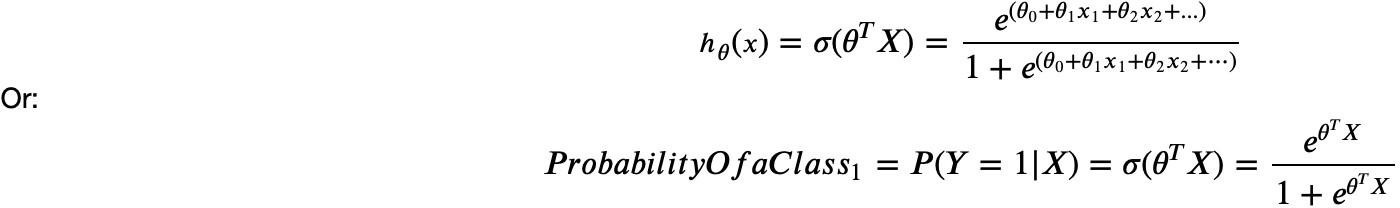
Unlike linear regression, which models a straight-line relationship, logistic regression fits an **S-shaped (sigmoid)** curve, using the **logistic (sigmoid) function**:

σ(θTx)=11+e−θTx\sigma(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}σ(θTx)=1+e−θTx1​

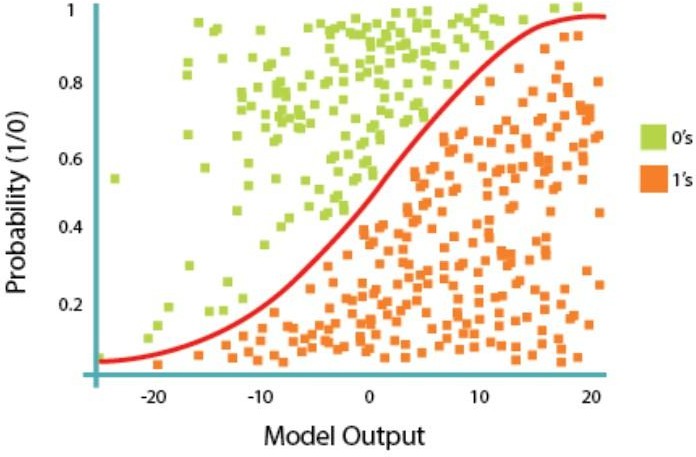
Where:

* θTx\theta^T xθTx is the linear combination of input features and weights,
* σ(⋅)\sigma(\cdot)σ(⋅) outputs a value between 0 and 1, representing a probability.

This **sigmoid function** ensures that the output is always between 0 and 1, making it interpretable as a probability (Figure 8).



In this equation, 𝜃!𝛸 is the regression result (the sum of the variables weighted by the coefficients), exp is the exponential function and 𝜎(𝜃!𝛸) is the sigmoid or logistic function, also called a logistic curve. It is a common "S" shape (sigmoid curve). So, briefly, logistic regression passes the input through the logistic/sigmoid but then treats the result as a probability *(Figure 8):*

**

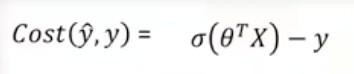
***Figure 8.*** *Logistic Regression; Model Output. (Source: Coursera).*

The objective of Logistic Regression algorithm is to find the best parameters θ, for ℎ"(𝑥) =

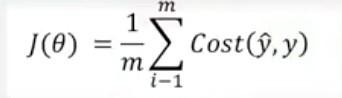
𝜎(𝜃!𝛸), in such a way that the model best predicts the class of each case. In other words, training a

logistic regression model aims to change the parameters of the model, so as to be the best estimation of the labels of the instances in the dataset. This leads to the next point of the cost function, using the derivative of the cost function we can find how to change the parameters to reduce the cost or rather the error. The cost function is the difference between the actual values of y and the model output predictions y hat (equal to 𝜎(𝜃!𝛸)). This is a general rule for most cost functions in machine learning.

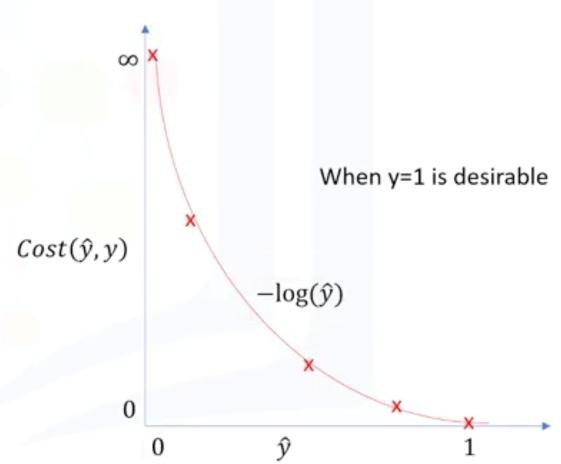
The cost function:



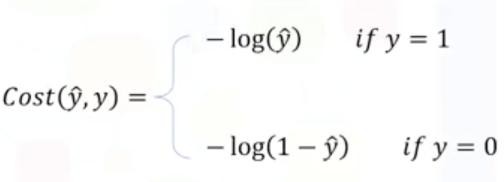
The cost function for all the samples in the training set:



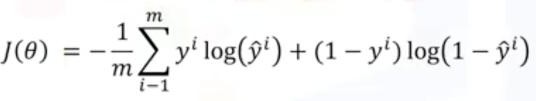
The way to find or set the best weights or parameters that minimize this cost function is by calculating the minimum point of this. Although we can find the minimum point using the derivative of a function, there's not an easy way to find the global minimum point for such an equation. The solution is to find another cost function instead, one which has the same behaviour but is easier to find its minimum point. The minus log function provides such a cost function *(Figure 9).*

**

***Figure 9.*** *The minus log function. The function returns zero if the outcome of the model is one. The cost is increasing as the outcome of the model gets further from one, and it is very large if the outcome of the model is close to zero. (Source: Coursera)* The cost can be calculated as:



After plugging this to the total cost function 𝐽(𝜃), we see the logistic regression cost function:



There are different optimization approaches to minimise the cost function, gradient descent is one of the most effective ones. Gradient descent, in this case, is a technique to use the derivative of a cost function to change the parameter values to minimize the cost or error. Gradient descent is like taking steps in the current direction of the slope of an error surface, and the learning rate is the length of the step or an additional control on how fast is the movement on this surface. To conclude, the training algorithm step-by-step is shown in *Table 3.*

### Decision Tree

**Decision trees** are intuitive and powerful classification tools that work by recursively partitioning the dataset into distinct subsets (called *nodes*), aiming to separate records into pure groups where all or most belong to the same class. At each **inner node**, a decision is made based on a test on an attribute, with each **branch** representing a possible outcome of the test. The process continues until a **leaf node** is reached, which assigns a class label.

**Construction of a Decision Tree**

The tree-building process involves:

1. **Selecting the best attribute** to split the data. This is done by evaluating how effectively an attribute separates the classes.
2. **Splitting the dataset** into branches based on the values of the chosen attribute.
3. **Recursively repeating** the process for each subset until stopping criteria are met (e.g., maximum depth, minimum samples at a node, or pure nodes).

The most predictive attribute is chosen at each step to maximize the effectiveness of the split, using **impurity measures**.

**Entropy and Information Gain**

**Entropy** measures the randomness or disorder in the dataset. It reflects the degree of class mixture in a node. A node with completely mixed classes has high entropy (close to 1), while a node with only one class has entropy 0.

Entropy for a node is calculated as:

H(S)=−∑i=1npilog⁡2(pi)H(S) = -\sum\_{i=1}^{n} p\_i \log\_2(p\_i)H(S)=−i=1∑n​pi​log2​(pi​)

Where pip\_ipi​ is the proportion of class iii in the dataset SSS.

To assess how useful an attribute is, we use **Information Gain**, defined as:

Information Gain=H(S)−∑j=1k∣Sj∣∣S∣H(Sj)\text{Information Gain} = H(S) - \sum\_{j=1}^{k} \frac{|S\_j|}{|S|} H(S\_j)Information Gain=H(S)−j=1∑k​∣S∣∣Sj​∣​H(Sj​)

Here, H(S)H(S)H(S) is the entropy before the split, and the second term is the weighted entropy after the split. A **higher information gain** indicates a better attribute for splitting.

**Recursive Partitioning**

The decision tree algorithm uses **recursive partitioning**, aiming to:

* **Minimize entropy** (increase purity),
* **Maximize information gain**,
* And select the best attributes near the root of the tree.

Each branch refines the classification by focusing on more specific subsets, continuing until nodes are either pure or no further improvement is possible.

### Multilayer Perceptron

A **Multilayer Perceptron (MLP)** is a class of feedforward artificial neural networks that can perform **prediction**, **classification**, and **novelty detection** through **regression-based learning**. It is a supervised learning algorithm and one of the most widely used models in deep learning for structured data.

An MLP consists of the following components:

* **Input Layer**: Takes the feature variables as inputs.
* **One or More Hidden Layers**: Each layer contains *neurons* (also called *perceptrons*), which apply an **activation function** to compute outputs from weighted inputs.
* **Output Layer**: Produces the final prediction. The choice of the output activation function (e.g., sigmoid, softmax, linear) depends on the type of task (binary classification, multi-class classification, regression).

The training process of an MLP involves:

* **Forward Propagation**: Inputs are passed through the network layer by layer until the output is computed.
* **Cost Function**: The network's prediction is evaluated using a **loss function** (typically the squared error for regression or cross-entropy for classification).
* **Backpropagation**: The network calculates the gradient of the loss function with respect to each weight using the **chain rule**.
* **Gradient Descent Optimization**: Weights are updated to minimize the loss. The **learning rate** determines how much to adjust the weights at each step.

The learning process is iterative and continues until the model converges or an early stopping condition is met (e.g., based on validation error).

**Hyperparameters**

MLPs are tuned using several **hyperparameters**, which include:

* **Learning rate**: Controls the step size during gradient descent.
* **Number of hidden layers and units**: Affects the capacity and expressiveness of the model.
* **Activation functions**: Common choices include ReLU, tanh, or sigmoid.
* **Early stopping**: Prevents overfitting by halting training when the validation error stops improving.
* **Batch size and number of epochs**: Influence how the model sees the data and for how long.

# Modelling

The various data mining techniques previously discussed—**Logistic Regression**, **Decision Tree**, and **Multilayer Perceptron (MLP)**—are applied in this section to develop classification models aimed at predicting **COVID-19 case trends and risk classifications** using public health indicators and demographic attributes.

In predictive modeling, achieving **high out-of-sample accuracy** is crucial, especially when dealing with public health data where model predictions can influence decision-making. To ensure robust evaluation, the **train-test split** method is employed. This involves dividing the dataset into two separate sets: a **training set** for model development and a **testing set** for evaluating the model's ability to generalize to unseen data.

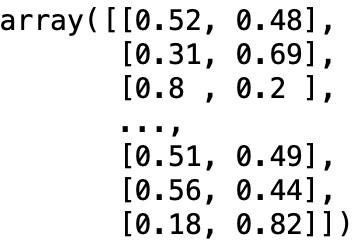
For this project, the **COVID-19 data tracker dataset**—which includes key variables such as infection rates, vaccination coverage, testing rates, and regional demographic indicators—has been used. After selecting the most relevant predictors through embedded feature selection, the dataset is split into **80% training data** and **20% testing/validation data**. This split remains consistent across all three models to ensure comparability and fairness in evaluation.

The **training data** is used to fit the models, enabling them to learn the underlying patterns and relationships that influence COVID-19 case trends and severity levels. The **testing data** is used to assess the performance of the models on new, unseen data. Evaluation metrics such as **accuracy**, **precision**, **recall**, and **F1-score** are used to quantify model performance.

The detailed experimental processes and model-specific evaluations are presented in the following sections.

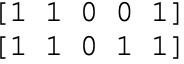
### Logistic Regression

The model is built using the function *LogisticRegression* from Scikit-learn package. The function implements the logistic regression and can use different numerical optimizers to find parameters, here I used the ‘liblinear’ solver and selected the parameter C equal to 0.01. This parameter indicates inverse of regularization strength which must be a positive float. Smaller values specify stronger regularization. The next step is to fit the model with the training set and then make predictions using the test set, the result is an array with predictions of the target variable. What is more, I utilised the *predict\_proba* method which returns the probability estimates for all classes which are ordered by the label of classes. So, the first column is the probability of class 1, P(Y=1|X), and the second column is the probability of class 0, P(Y=0|X), the outcome is shown below;

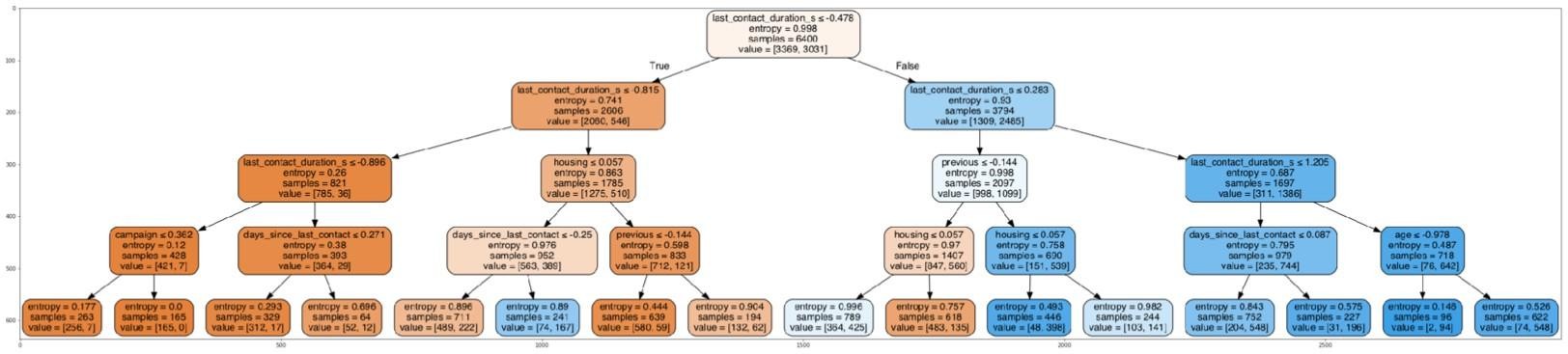


### Decision Tree

The model is built by creating an instance of the DecisionTreeClassifier called *depositTree*. Inside of the classifier, the Criterion is specified as ‘entropy’ so we can see the information gain of each node and Max Depth equals to four. Next, fitting the data with the training feature matrix X\_train and training response vector y\_train that have been created previously with *train\_test\_split* method (mentioned previously). The next step is to make predictions on the testing dataset, i.e. using the trained decision tree to predict the class of an unknown customer, or to find out - given customer’s characteristics if subscribes the term deposit. To visually compare the predictions made to the actual values I print the predictions mentioned and the y\_test set. For example, a screenshot of this;



Lastly, I have further explored the decision tree (*Figure 9) by its visualization*.



***Figure 9.*** *Visualization of the Decision Tree, depicting the nodes and information gain on each leaf.*

### Multilayer Perceptron

The model I trained last is an MLP Classifier. I have used the same attributes as in the previous models which have been noted in the data preparation section. Again, 20 percent of the dataset has been assigned to the test set that will help us validate the accuracy of the model.

The hyperparameters that selected include;

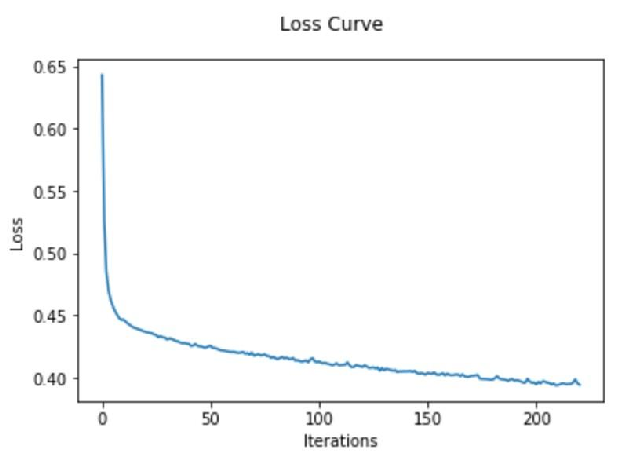
***Hidden Layer Sizes to (32, 100)*** The number of neurons for each hidden layer and the number of hidden

layers

|  |  |
| --- | --- |
| ***Max Iterations to 500*** | The number of iterations of the solver |
| ***Alpha equal to 0.0001*** | Regularization parameter |
| ***Activation to Relu*** | The activation function for the hidden layers |
| ***Solver to Adam*** | The solver for weight optimization |
| ***Verbose to True*** | Whether to print progress messages to stdout |
| ***Shuffe to True*** | Boolean that defines if to shufle the data at each  iteration |
| ***Random State to 21*** | the seed used by the random number generator |

Next, I fitted the model with the training set and made predictions on the testing dataset which

revealed the loss according to its iteration time. The models’ loss curve is depicted in *Figure 10.*

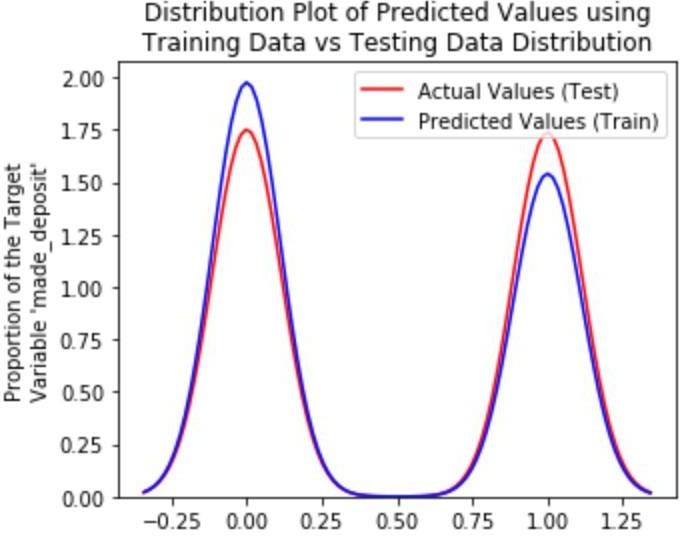
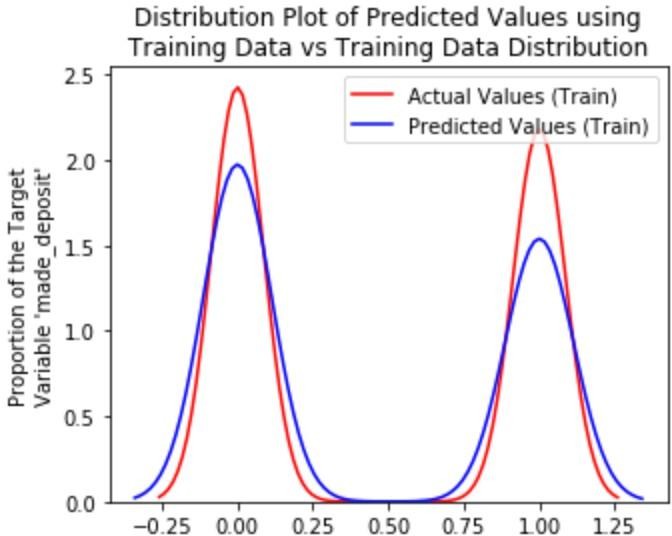
**

***Figure 10.*** *The loss curve of the MLP Classifier model*.

# Results and Errors

The models discussed in the previous sections can help us explore the connection between selected features and predict the target. This section lists the results and compares the accuracies of the predictions made by the models. Furthermore, the representation of the ROC curve of each model will be depicted here. **Logistic Regression**

The distribution plot *(Figure 11)* shows how the model is doing in learning from the training data set*.* When the model generates new data from the testing dataset we see *(Figure 12)* the distribution of the predicted values is slightly different from the actual target values, however, it seems very close to them. Comparing the figures, the distribution of the data in figure 12 is much better at fitting the data, however, there are small ranges that the opposite is observed.

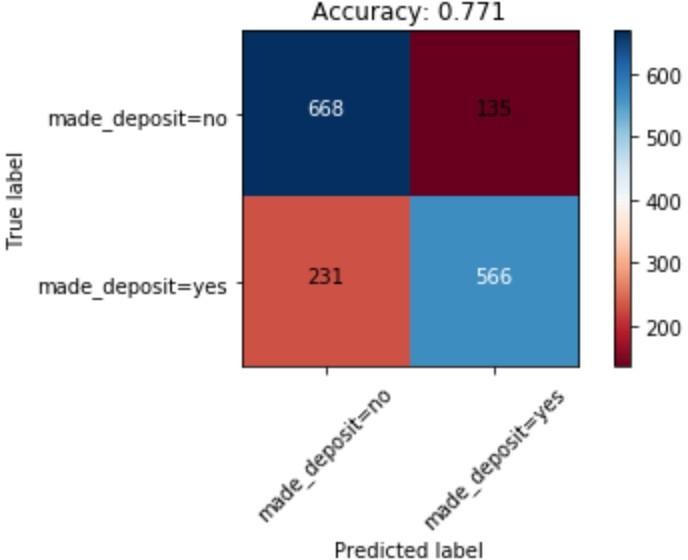


***Figure 11.*** *Logistic Regression****;*** *Plot of predicted values* ***Figure 12.*** *Logistic Regression; Plot of predicted values using the training data compared to the training data. using the training data compared to the testing data.*

It turns out that the test data sometimes referred to as the out of sample data is a much better measure of how well your model performs in the real world. One reason for this is overfitting. Overfitting occurs when the model fits the noise, not the underlying process. Therefore when testing the model using the test set, the model does not perform as well as it is modelling noise, not the underlying process that generated the relationship.

Regarding the accuracy of the model, I have utilized the Jaccard index and Log loss metrics, the Jaccard similarity coefficient found equal to 0.771 and the log loss equal to 0.492. Ideal classifiers have progressively smaller values of log loss, meaning lower log loss has better accuracy.

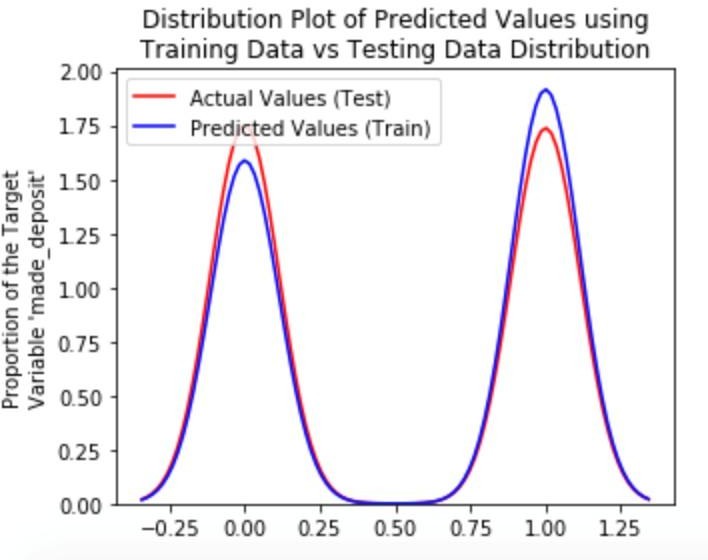
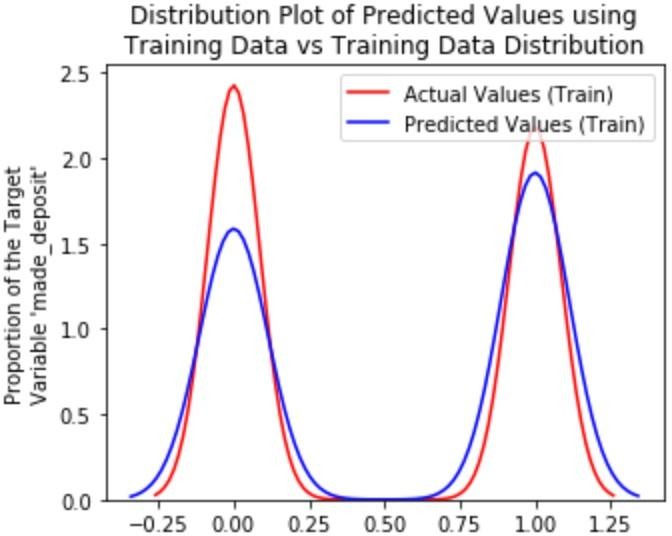
Furthermore, the level of accuracy and errors made by the model is shown in the logistic regression confusion matrix *(Figure 13).*



***Figure 13.*** *Logistic Regression Confusion Matrix.*

### Decision Tree

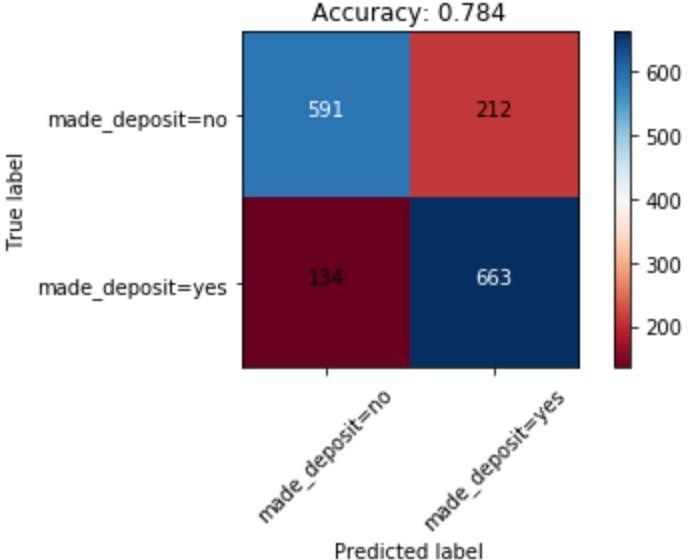
The distribution plots, firstly the *Figure 14* depicts how the model is doing in learning from the training data set, and secondly*, Figure 15* shows how the model performs when encounters new data from the testing dataset.



***Figure 14.*** *Decision Tree; Plot of predicted values using the* ***Figure 15.*** *Decision Tree; Plot of predicted values using the training data compared to the training data. training data compared to the test data.*

Regarding the accuracy of the model, I have utilized the Jaccard index, F1-score metrics as well as the accuracy score method for comparison purposes. All of them returned the same value of *0.784* approximately.

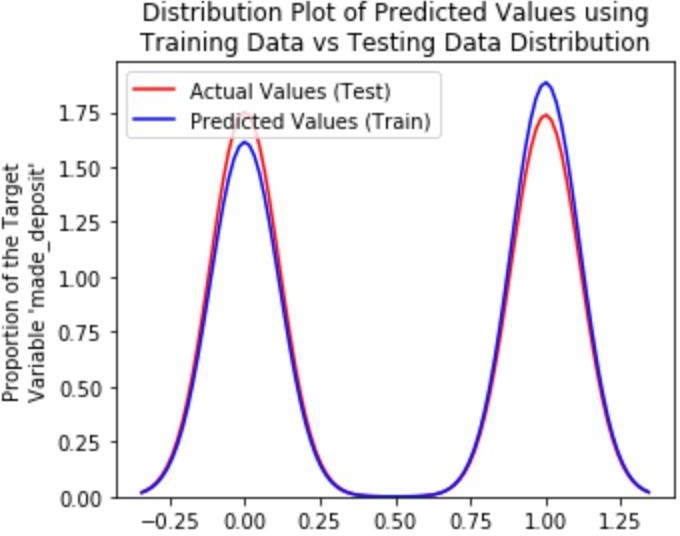
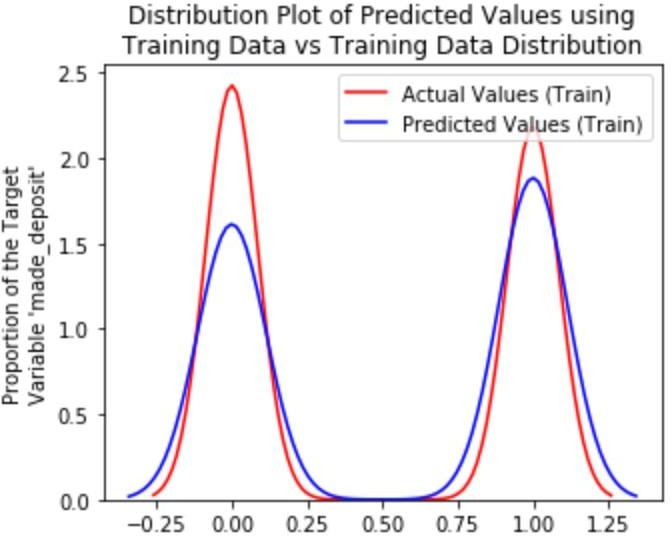
The confusion matrix of the model is shown below *(Figure 16).*



***Figure 16.*** *Decision Tree Confusion Matrix.*

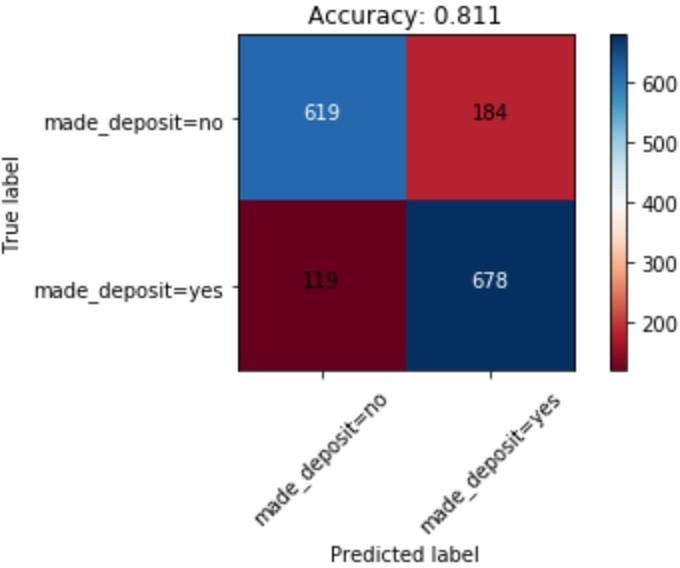
### Multilayer Perceptron

Similarly, The distribution plots, firstly the figure 17 depicts how the model is doing in learning from the training data set, and secondly, Figure 18 shows how the model performs when encounters new data from the testing dataset.



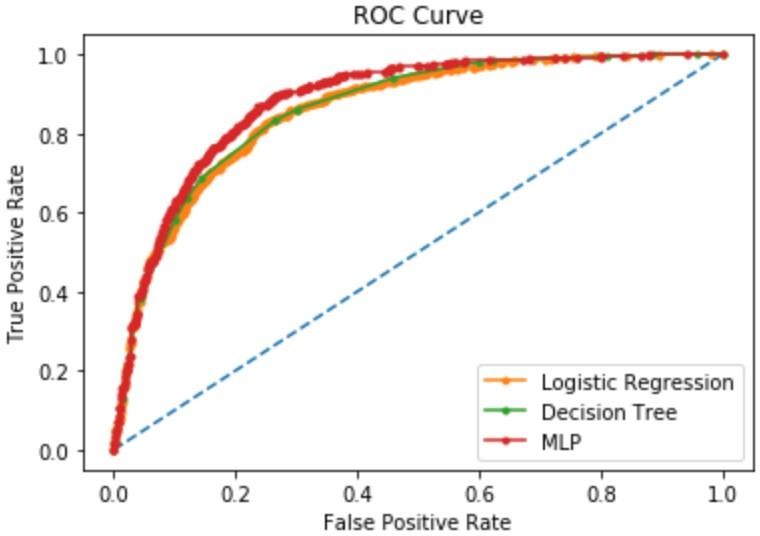
***Figure 17.*** *MLP; Plot of predicted values using the training* ***Figure 18.*** *MLP; Plot of predicted value using the training data compared to the training data. data compared to the test data.*

As far as the accuracy of the model, I have utilized the accuracy score method. The accuracy of the MLP model equals to 0.810. A visualization of the confusion matrix of the model is shown below *(Figure 19).*



***Figure 19.*** *Multilayer Perceptron Confusion Matrix.*

Receiver Operating Characteristic (ROC) curve is a tool which supports the explanation of probabilistic prediction for binary classification modelling, and here its use is rational since we have roughly equal numbers of instances for each class. The curves show the true positive and false positive rate for every probability threshold and reveal that the most skillful model is the MLP, bowing up to the top left of the plot *(Figure 20), with AUC performance equal to 0.882*.



***Figure 20.*** *ROC Curves of the Logistic Regression , Decision Tree and Multilayer Perceptron models. Logistic Regression: AUC=0.860, Decision Tree: AUC=0.861, MLP: ROC AUC=0.882.*

It is important to mention that utilizing a different set of variables or even different classification ML models would give different results and accuracy levels. Moreover, implementing the models with different hyperparameters would probably help in obtaining different results towards the improvement of the predictions. For example, regularization parameters have been used, which is a

technique able to solve the overfitting problem in machine learning models. Numerous of alteration in the process I followed would have a different positive or negative effect.

Concluding, overall and all three models taken into consideration, the results suggest that the MLP has better performance than the other classification models since its accuracy is higher.

Consequently, with the current data-processing, variables selection, ML models and hyperparameters, the proposed model in order to predict whether a customer will subscribe to the term deposit is the Multilayer Perceptron.

## REFERENCES

1. R. J. Robles, R. Alcaria, D. M. de Andrés, M. N. de la Cruz, R. Calero, S. Iglesias, and M. Lopez, “An IoT based reference architecture for smart water management processes,” *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, vol. 6, no. 1, pp. 4–23, 2015.
2. S. W. Lee, S. Sarp, D. J. Jeon, and J. H. Kim, “Smart water grid: The future water management platform,” *Desalination Water Treat.*, vol. 55, no. 2, pp. 339–346, 2015.
3. A. K. Jain, M. N. Murty, and P. J. Flynn, “Data clustering: A review,” *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, 1999.
4. C. Cortes and V. Vapnik, “Support-vector networks,” *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.
5. J. R. Quinlan, “Induction of decision trees,” *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.
6. Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
7. L. Breiman, “Random forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
8. T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. New York, NY: Springer, 2009.
9. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016.
10. G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
11. F. Pedregosa et al., “Scikit-learn: Machine learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
12. M. Kuhn and K. Johnson, *Applied Predictive Modeling*. New York, NY: Springer, 2013.
13. D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,”

*arXiv:1412.6980*, 2014.

1. S. Raschka, *Python Machine Learning*. Birmingham, UK: Packt Publishing, 2015.
2. C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY: Springer, 2006.
3. T. Mitchell, *Machine Learning*. New York, NY: McGraw-Hill, 1997.
4. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
5. A. Ng, “Machine learning yearning,” *DeepLearning.ai*, 2018. [Online]. Available: <https://www.deeplearning.ai/machine-learning-yearning/>
6. J. D. Kelleher, B. Mac Namee, and A. D’Arcy, *Fundamentals of Machine Learning for Predictive Data Analytics*. Cambridge, MA: MIT Press, 2015.
7. G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*. New York, NY: Springer, 2013.