Attached3

Graduation Work/Thesis Final Report

2022 year 2nd semester

Title : **Smartphone-Based Food Assessment using Semantic Ontology and Deep Learning**

Student Name (student no.)

1. Aizat Hamizuddin Bin Azlan (2019313858)
2. **Amirah Binti Ahmad Nadzri (2019313857)**
3. Fatdzirul Izzat Bin Abdul Radzi (2019313856)
4. Nurlaili Binti Zainal Abidin (2019313854)

2022.11.09.

Advisor: Professor Tamer Abuhmed Sign [illegible]

Plan(10)	Theme(20)	Concept(20)	Detail(30)	Report(20)	Total(100)
10	20	18	28	19	95

* The score will be filled by the advisor.

Abstract

In this article, a research project on a mobile dietary preference system that performs its own inferencing rules in its own knowledge base is discussed. Personal food diet preference has been embedded in our culture through the existence of religions, beliefs, and moral values. Currently, there is no existing application or tool available that could directly assist the public in choosing and categorizing commercial consumable products that cater to their personal food preference. This might be difficult for consumers with diet preferences such as veganism and vegetarianism when purchasing commercialized goods since these products normally consist of long lists of various ingredients. To help these consumers with specific diet restrictions, we created this project which aims to create a mobile application that guides consumers globally in choosing personalized products that suit and comply with their dietary preferences. By utilizing Optical Character Recognition (OCR) in Flutter, and Web Scraping with the assistance of Selenium and Machine Learning-Classification, the mobile app would read text, query a server of ontologies, and then classify them based on several major food constrictions that are practised. After scanning the ingredients on the product, the final result of whether it fits the consumer's diet preferences will show up. The application would serve as a solution and a convenience to the public in ensuring that their preferred diet is obliged.

1. Introduction

In this 21st century, there is no doubt that human culture and biology have evolved vastly compared to the past, and will continue so in the future. This is especially evident in the variety of our food and its classifications which not only originate from a lot of countries, races, and religions but also intertwine between them as humans are free to travel anywhere. Nowadays, dietary restrictions are not only tied to health restrictions but also have become famously known for being tied to human physiology, values, or sometimes even by choice. Among the common dietary restrictions that are quite known these days consist of lactose intolerance, gluten intolerance or sensitivity, vegetarianism, veganism, and shellfish intolerance [1].

However, this can sometimes become a problem for consumers in terms of purchasing commercial goods as factory-made goods as they are sometimes not direct in informing the consumers regarding the ingredients that are used in the products. For instance, people who are lactose-intolerant, vegan, and vegetarian often find it inconvenient to shop for goods that are in line with their personal values. There might be countries that are very suited to a certain type of diet but there are also countries that are not. For example, Muslim-centric countries have a lot of halal-certified commercial goods and would be comfortable and easy for people who consume halal food to purchase them, while in western countries, it would not be as easy for those people to purchase halal-certified goods. Halal-certified products are products that are permissible or acceptable in accordance with Islamic law [2].

In modern days, the list of ingredients stated on food products is now substituted with scientific terms and codes to further cast a filter even though corporations are well informed on how they should be transparent with their consumers. The possible cause of this may be because directly revealing ingredients might decrease the size of their current niche. Currently, some applications can aid consumers with the most common dietary restriction, such as halal products, vegan products, gluten-free products, and keto products.

However, these applications are restricted and only limited to a specific dietary restriction, location, and available product database. This would force the consumer to use a different application for extra concerns. An application for people that have more than one category of dietary restriction is considered a need in this current society as they can scan the ingredients, which the application then informs the users on whether they should or should not be concerned about the products before buying. In addition to creating a single application with multiple purposes for users without having to use different applications for different needs, we are also helping users with dietary restrictions to be able to search for commercialized goods that are suitable for them to consume following their personal diet.

The existence of this project would not only benefit users on a daily basis, but it would also be useful when users are visiting a foreign country where the local population does not value the same restrictions that the user does. With the number of students studying abroad increasing and the trend of the Digital Nomad who is someone that is location-independent and uses technology to perform their job, living a nomadic lifestyle is getting more and more attention, this could be useful for users who are living whether permanently or not in another country or even those who enjoy travelling [3]. Other than that, it may also help people who are still exploring their preferred food diet and assist them in their journey.

In this project, we aim to directly assist people all over the world in further analyzing their preferred goods and obtaining more information on whether the goods being sold are aligned with their dietary restrictions by utilizing existing ontology on food classifications, and multiple Machine Learning models in Artificial Intelligence. Ontologies can be used to index and access semi-structured information sources, and domain ontologies are required by Internet search engines to organize information and guide search processes [4]. By searching through a wide range of ontology in the provided designated server, the application can serve and deliver data to the machine learning classification model and immediately make the users know the desired information about whether their diet is listed among the ones that the product complies with.

Additional features that adhere to the theory of User Experience and User Interface will also be implemented in this project, ensuring the application's global accessibility and marketability. An application must not only perform its function effectively but must also be user-friendly, emphasizing the importance of designing the application in accordance with what global users seek. Therefore, the goal of this project is not only to serve its main purpose which is assisting users regarding whether the product fits their dietary preferences, but also an application where it can be easily understood and straightforward for the user.

Our project’s mobile application offers a simple yet functional feature and design to ensure the users’ needs are fulfilled with little to no complications while using the app. The app consists of a user login system where users can sign-up and login into their accounts. Users can also just continue as guests and use the app easily if they personally do not want to create an account. A demo gif pop-up will appear every time a user wants to use the app in order to offer guidance on how to use the app. A choice is given to the user to either use the device’s camera or select a picture from the device’s gallery. After choosing an image, the user can crop the image to specifically take only the ingredients label on a product. Then, the app will extract the list of ingredients from the image in addition to the user being able to edit or delete any ingredients that are not needed or have minor inaccuracies such as wrong spelling. The app will then produce an output to the user whether the scanned product is suitable for their dietary preference group.

The framework of the project consists of the mobile interface, database, ontology, and machine learning model. Users scan the food ingredient list from the product. Then, the mobile application will tokenize the list of ingredients which will be given to the database to query the ontology. The ontology returns a list of categories based on food preferences. In this project, we obtained a dataset from Kaggle containing the name of food products and their list of ingredients.

The rest of this paper is planned as follows: Second section presents the related work that we found based on our research. It explains the purpose of their work, what had been done, the outcomes of their experiment, and the comparison and difference between their overall work and ours. The third section presents the proposal of our work that explains the methods that we use for our project, how the concept of those methods works, how we integrated them into our project, and what frameworks were utilized in technical detail. The fourth section presents the experiment and evaluation that have been done on the dataset, results of the metrics, and discussion based on our observation. The Sixth section is dedicated to the conclusion of this project.

2. Related Research

A project by a team that is led by Kim Minseob, which is an application that is being constantly developed called MUFKO (Muslim Friendly Korea), uses Optical Character Recognition (OCR), translation Application Programming Interface (API), and a huge database to identify the trigger ingredients that make a product non-Halal and not suitable Muslim consumption [5]. The project scans the ingredients on the label using Kakao OCR API and translates them using Papago API while our project uses OCR In Flutter. The end result will categorize the ingredients that fall under the trigger words. However, the current project only caters to religious-related diets, specifically the Halal diet. Meanwhile, the main contribution of our work to this issue is by catering to a wider range of diet preferences and not relying only on databases to identify food categories. Usage of ontology would ensure different and confusing terms used in the food industry are well categorized thus overcoming the limitation of the project that relies more on translation.

In February 2018, a research was published regarding the application of Deep Neural Network (DNN) models in dealing with data categorization for food diet preferences. Then, the categorization of data allows for an automated recommendation of grocery products [6]. The system framework is made up of four main components, which are the use of word embedding, a DNN model for product categorization, a decision recommendation model for analyzing the DNN model, and lastly, an operational state machine for controlling the state and retraining the models whenever there are updates on the training data. Opposite to our work, the DNN model suggests grocery products that use a lot of resources and are heavy on the database and training data. The contribution of our work solves this issue where no database on products is being used and rather informs users on whether the product is aligned with their food preferences.

Coskun et al. conducted a research where they used a physical personalized food allergen testing platform, called iTube, which is connected to a cellphone then captures images and automatically analyzes colourimetric assays performed in test tubes toward sensitive and specific detection of allergens in food samples [7]. iTube is a smartphone-based digital reader that will need to be attached to the back of the cell phone. There will be an app connected to this device that will show the number of allergens existing in a product. In regards to this research, our project tackles the limitation of the research that uses the niche of mandatory dietary restrictions that are caused by medical issues in terms of allergies and body intolerances. Our work is more open to dietary preferences and focuses our niche on certain chosen lifestyle diets. In addition to that, this project used an external device to scan the products while our project aims to scan the product using the existing smartphone camera.

Meanwhile, Ertuğrul. had done a research on Safety Food Consumption Mobile System (SFCMS) through Semantic Web technology [8]. This system is said to be an ontology-based software application in which its semantic rules are created in OWL 2.0 by using Protégé editor2 in the form of SWRL3 (Semantic Web Rule Language). Similar to our project, this research utilizes food ontology in order to get suggestions on the suitable and appropriate contents of the food product on the market shelves. The queried products will then return the details of the ingredients and nutrients.

In another paper by Abdullah et al., they did a research on an Android-based mobile application named Food Code Breaker (FCB) [9]. This app aids in identifying and verifying whether the status of the food product is halal, vegan and contains allergy or not. While our project scans the list of the ingredients on the product, FCB scans the ‘E numbers’ in the ingredients list which are codes used to replace common names of particular food additives or chemicals. The similar point of their project and ours is that both use the Optical Character Recognition (OCR) system. However, FCB outputs the translation of the ingredients into the desired language, which differs from the output of our project.

Eatable, a mobile application done on a research by Sabina, is an application that helps consumers in searching for allergen-free food [10]. Eatable checks the consumer’s location and creates records in its database by listing nearby stores that have the products that are safe. This application utilizes Google Web Services API and Label API third-party services to fetch the consumer’s location and also the product information. Similar to our project, Eatable has a check-product feature to check the product’s safety, but it differs from ours as Eatable has a feature that can check where the product is available according to the user’s location.

In another research done by Junaini et al., they worked on an application that scans the barcode on the product, which then will return the output to the consumer whether the product is halal or not [11]. Compared to our project, this research focused solely on verifying whether the product is halal according to the Department of Islamic Development (JAKIM) in Malaysia. Meanwhile, our project focuses on a lot of other categories such as veganism, and vegetarianism.

Abao et al. worked on a research in creating FoodGo, a mobile application that uses situated analytics to show product information to help consumers in making food choices while grocery shopping [12]. FoodGo prompts the users to scan the barcode using the mobile camera which then the application will print an output of the product information. Compared to our project, FoodGo scans the barcode of the product while ours scan the list of the ingredients. FoodGo outputs the nutritional and food information of the product, while our project displays an output of whether it is suitable for the user to consume based on their food preferences.

In a research by Dunford et al., they created a mobile application named FoodSwitch that helps consumers track the nutritional composition of food products in Australia [13]. The application allows consumers to scan the barcode on the packaging which then displays the information regarding nutrition back to the consumers. Their project scans the barcode of the product while ours prompt the users to scan the list of ingredients. FoodSwitch also only displays the output of the detailed nutrition information of the product, which is different to ours as mentioned previously.

Shari et al. had done a research on a mobile application that recommends suitable food for babies with allergies. The system of the application aids the parents and caretakers of the babies by receiving information on the babies' allergies and outputs food that is safe to be consumed by the babies. Compared to ours, the target of this project is more specific, while ours is broader. However, the concept of both our projects is quite similar which is to help people with allergies and food preferences [14].

Table 1 below shows the comparison between the aforementioned works to ours in terms of whether they offer database usage, dataset availability, dataset size, ontology, type of ontology, machine learning model, word embedding, availability on mobile, account user creation, camera access, photo gallery access, scan feature availability, optical character recognition, dietary restrictions (health/lifestyle), religious dietary restrictions, food allergen, personalized diet, and nutritional information.

Research Paper	Database Usage	Dataset availabiliy	Dataset Size	Ontology	Type of Ontology	Machine Learning Model	Word Embedding	Available on mobile	Account User Creation	Camera Access	Photo Gallery Access	Scan Feature Availability	Optical Character Recognition	Based on Dietery Restrictions (Health/Lifestyle)	Based on Religious Dietery Restrictions	Based on Food Allergen	Offers Personalized Diet	Shows Nutrional Information
[5]	✓	✖	✖	✖	✖	✖	✖	✓	✓	✓	✖	✓	Kakao OCR API	✖	✓	✖	✖	✖
[6]	✓	✓	942, 9126, 2012	✖	✖	LSTM-RNN, CNN	✓	✖	✖	✖	✖	✖	✖	✖	✖	✖	✓	✖
[7]	✓	✖	✖	✖	✖	✖	✖	✓	✖	✓	✖	✓	✖	✖	✖	✓	✖	✖
[8]	✓	✖	✖	✓	undisclosed	✖	✖	✓	✓	✓	✖	✓	✖	✓	✖	✖	✓	✖
[9]	✓	✖	✖	✓	undisclosed	✖	✖	✓	✖	✓	✓	✓	undisclosed	✓	✓	✓	✖	✖
[10]	✓	✖	✖	✖	✖	✖	✖	✓	✓	✓	✖	✓	✖	✖	✖	✓	✖	✖
[11]	✓	✖	✖	✖	✖	✖	✖	✓	✖	✓	✓	✓	✖	✖	✓	✖	✖	✖
[12]	✓	✖	✖	✖	✖	✖	✖	✖	✖	✓	✖	✓	✖	✖	✖	✖	✖	✓
[13]	✓	✓	17,000	✖	✖	✖	✖	✓	✖	✓	✖	✓	✖	✖	✖	✖	✖	✓
[14]	✓	✖	✖	✖	✖	✖	✖	✓	✓	✖	✖	✖	✖	✓	✖	✓	✓	✖
Ours	✓	✓	13,020	✓	FOODON, CHEBI, NCIT	RNN, MLP	✓	✓	✓	✓	✓	✓	Flutter OCR	✓	✖	✓	✖	✖

Table 1. Comparison between related works.

3. Introduction of Proposal Work

In this section, we will be discussing the methods that we use for our project, how the concept of those methods works, how we integrated them into our project, and how they were utilized into our project in technical detail. The explanation starts with the application building from the front-end side, the utilization of RESTful APIs in the back-end side, the querying of ontology, and lastly, the classification of the ingredients by using machine learning models.

3.1 Front-end

We mainly used **Flutter** for our app development**.** The features of Flutter shorten the application development time and convenience in user interface development all while maintaining a clear architecture and excellent control over elements [15]. Flutter utilizes Dart, which increases the readability of the core code of the application and allows the separation of logic and the UI, thus ensuring a better debugging process [15]. Flutter also offers a better performance in terms of certain queuing operations, sorting, filtering, serialization, or operations on the files [16]. While Flutter may have its own limitations, the advantages of Flutter such as better utilization of the CPU usage must not be turned a blind eye to [16]. In the early development, a flow diagram was made using Figma for visualizing how the app would flow through what features to help ease the programming and building process of the user interface.

This app consists of the login, signup and user guide features. When launching the app, the first page would be the 'Login Page' as depicted in **Figure 1**. The users can insert their username and password to log in if their account has already been registered in the database but if not, then the user can create a new account by clicking the ‘Sign up’ button which will redirect the users to the ‘Sign up Page’ where they have to enter their information such as their email, username, first name, last name and password as shown in **Figure 2**.

These pieces of information will be sent to the database for storage. Passwords are hashed with the SHA-256 algorithm to keep the password hidden and safe from any potential security threat [17]. The users also have the choice to skip the registration process and just continue as a guest to use the app function without having to log in. After logging in, a pop-up will appear to show a user guide or demonstration on how to use the camera and cropping feature. **Figure 3** shows the said demonstration pop-up.

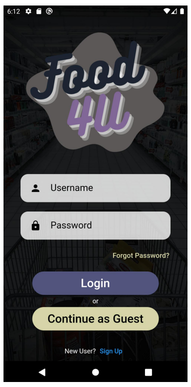


Figure 1. Login Page.

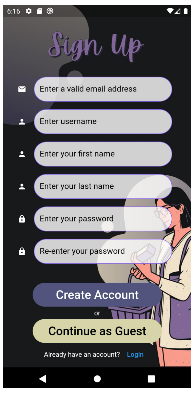


Figure 2. Signup Page.

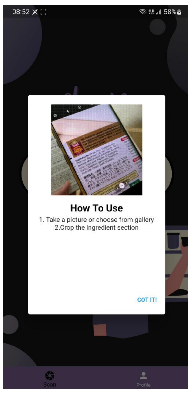

How To Use
1. Take a picture or choose from gallery
2.Crop the ingredient section
GOT IT!
Figure 3. Demonstration pop-up.

Figure 4 shows the main page which is the home page of the app containing 2 tabs, the camera feature and the user profile feature. The camera feature offers the user to choose between taking a picture using the camera or picking an image from their device gallery. **Figure 5** depicts the user profile page where the user’s preference settings and user information such as username and full name can be found. A total of 7 choices of dietary preferences including ‘No Preference’ can be chosen according to the user’s needs.

When **extracting the text from the image,** we used the **Optical Character Recognition in Flutter plugin.** After choosing the image of the ingredients of a product, a cropping mechanism will allow the user to crop out the image to specifically show only the list of ingredients as seen in **Figure 6** to ensure the accuracy of the simple_OCR_plugin which is a compact Optical Character Recognition for flutter apps backed by Google's ML-Kit library is used [18].

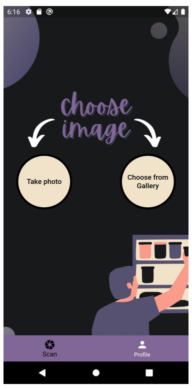
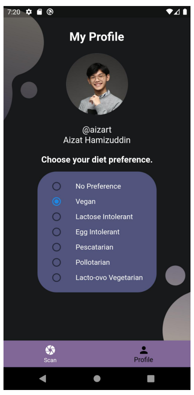


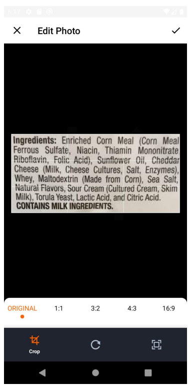
Figure 4. Home Page.



Choose your diet preference.

- No Preference
- [✓ Vegan]
- Lactose Intolerant
- Egg Intolerant
- Pescatarian
- Pollotarian
- Lacto-ovo Vegetarian

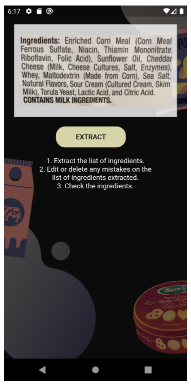
Figure 5. User Profile Page.



Ingredients: Enriched Corn Meal (Corn Meal Ferrous Sulfate, Niacin, Thiamin Mononitrate Riboflavin, Folic Acid), Sunflower Oil, Cheddar Cheese (Milk, Cheese Cultures, Salt, Enzymes), Whey, Maltodextrin (Made from Corn), Sea Salt, Natural Flavors, Sour Cream (Cultured Cream, Skim Milk), Torula Yeast, Lactic Acid, and Citric Acid. **CONTAINS MILK INGREDIENTS.**

Figure 6. Image Cropper.

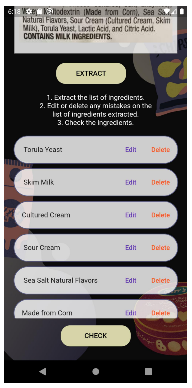
Once cropped, the image will be displayed on the page alongside instructions as seen in **Figure 7**. When the 'EXTRACT' button is pressed, the simple OCR plugin will perform OCR on the image to extract the text as a string in JSON format. The string is decoded to extract only the ingredients of the JSON into another string. The list of ingredients then undergoes preprocessing to ensure they are as accurate as they are according to the original list of ingredients in terms of spelling and spaces. The list can still be edited and deleted by the user if there is any incorrect input or spelling produced by the simple OCR plugin as shown in **Figure 8** and **Figure 9**. Once the list is finalized, the ingredients are then sent to the server side by using the HTTP GET method when the 'CHECK' button is pressed.



Ingredients: Enriched Corn Meal (Corn Meal Ferrous Sulfate, Niacin, Thiamin Mononitrate Riboflavin, Folic Acid), Sunflower Oil, Cheddar Cheese (Milk, Cheese Cultures, Salt, Enzymes), Whey, Maltodextrin (Made from Corn), Sea Salt, Natural Flavors, Sour Cream (Cultured Cream, Skim Milk), Torula Yeast, Lactic Acid, and Citric Acid. **CONTAINS MILK INGREDIENTS.**

1. Extract the list of ingredients.
2. Edit or delete any mistakes on the list of ingredients extracted.
3. Check the ingredients.

Figure 7. Before extracting the ingredients.



1. Extract the list of ingredients.
2. Edit or delete any mistakes on the list of ingredients extracted.
3. Check the ingredients.

Torula Yeast	Edit	Delete
Skim Milk	Edit	Delete
Cultured Cream	Edit	Delete
Sour Cream	Edit	Delete
Sea Salt Natural Flavors	Edit	Delete
Made from Corn	Edit	Delete

Figure 8. Ingredients extracted by OCR.

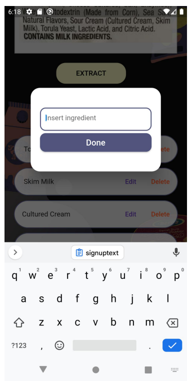


Figure 9. Window pop-up for editing.

After pressing the 'CHECK' button, a loading indicator as reflected in **Figure 10** will show up while waiting for the server side to return the results. Once results are retrieved, an image of a green-coloured thumbs-up icon or red-coloured thumbs down will be displayed alongside the results on which group dietary group is suitable to consume the food product as seen in **Figure 11**. Pressing the "Return to Home" button will redirect the user back to the home page. In cases where the application is not able to connect to the server side, a pop-up will show up to notify the user to try and send the ingredients again as depicted in **Figure 12**.

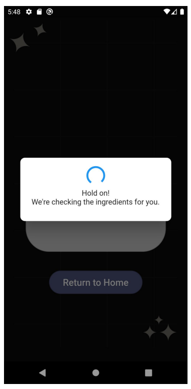


Figure 10. Loading indicator.

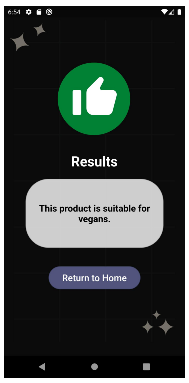


Figure 11. Results Page.

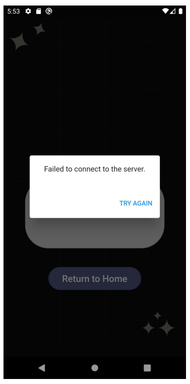


Figure 12. Failed to connect to server.

3.2 Back-end

Django is used in this project in order to connect the front end to the database, ontology, and machine learning. Django is a high-level Python web framework that provides data for the front end on request by the users and also the application that channels it. It also contains the database, which organizes all of the information created from the front-end side, such as account details, and also organizes pre-existing data used for machine learning models. Django is written in the language Python, which will be easy to work with as it is quite powerful and is used in the implementation of scientific computations and high-level Artificial Intelligence, which is also fitting to our project as we will implement deep learning for the classification of dietary preferences [19]. Django comes with the **Django REST Framework** which is very useful as APIs are very much needed for the nature of this project. Django Rest Framework allows developers to create RESTful APIs, which is a way to transfer information between an interface and a database in a simple way [20].

For the register and login parts that the user fills in from the front-end, POST API requests are used since the user will send data about their account details to the server to create, fetch, or verify the data in the database. In addition, GET API requests are also used to process and fetch the classifications of the dietary references from the ontology and machine learning models, to the front-end as the end result for the users.

As for the **composition** of files in Django, it is composed of ‘project’ and ‘app’, which have their own functionalities. A project in Django is a python package that represents the whole web application, containing the configuration and settings related to the entire website [21]. Meanwhile, an app is a sub-module of a project, and we can also have multiple apps in it that can be used to implement some other and separate functionalities [21]. In both projects and apps, there are important files that are pertinent to making the connection between the front-end, REST

APIs, and the database work. In this project, ‘backendproject’ is the project, and ‘accounts’ is the added app. ‘backendproject’ is where the main configurations are, and the code for the web scraping of the ontology, and machine learning model are located there. ‘accounts’ is the app where the register and login API are stored. It also stores all of the users’ details, such as username, password, first name, last name, and dietary preferences in the database.

We used **PythonAnywhere** in order to host the aforementioned server that has been made with Django, and also to make it available online. PythonAnywhere is an online integrated development environment (IDE) and web hosting service (Platform as a service) and it eases developers to create and run Python programs in the cloud [22]. Therefore, it is suitable for this project as it is compatible to be used with Django as it uses Python. This platform hosts our application and is the backbone of sending RESTful APIs from the front-end to the back-end, including ontologies and machine learning models, and vice versa.

3.3 Ontology

Ontology or query expansion, aims to automatically transform a user query into one that is more suitable for information retrieval. It is typically conceived of as a recall-based strategy [23]. Ontologies appear to be a good direction for query growth going forward, by rephrasing the user's question using context identification and disambiguation, they increase the accuracy of fuzzy information search [24].

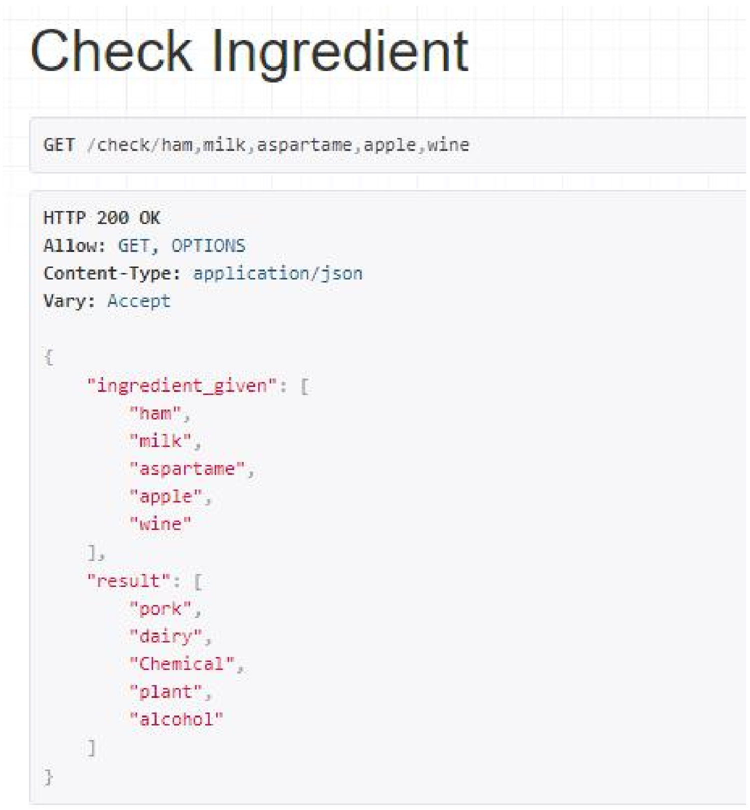
The ontology that we use has an error-prone flaw, which is that it cannot differentiate words with more than one meaning. Before sending any keyword (string) to be queried with the ontology to be crawled through, a list of Python codes will scan through our customized trigger word list to increase the accuracy of the categorization. A well-selected trigger word list is essential to the query as the project admits that using an openly available ontology that was also made by humans is prone to flaws and makes it vulnerable to errors. Doing so will ensure that any words that may relate to a certain category could be recognized and be sorted beforehand, thus improving the accuracy of the query. If none of the queried strings matched the list, the keyword will be sent to Ontobee to be crawled through all three main ontologies which are FOODON (main ontology), NCIT (supplementary ontology), and CHEBI (ontology for chemical substance).

By utilizing the Selenium library for queries, Ontobee will crawl through the ontologies sequentially and return the 10 nearest parent classes to be checked through the trigger word list again. During this process, Ontobee utilized RDF and OWL language to call the parent class of the current keyword class repeatedly until the ten parent class is obtained. In order to ensure that the website can be scraped in the long run, the Selenium library is being used in our backend through PythonAnywhere due to the ability of the library to run dynamic websites using a headless browser. Different from XHTML parser libraries that are used for static websites, the Selenium library suits better for the project since the website utilizes the ontology using OWL language to further call the parent class, thus making the website dynamic.

The reason for the project approach is due to the low to none possibility of obtaining each ontology core for free, and creating a new list of RDF and OWL code is a waste of resources. Other than that, the project also utilizes the Chromium Chromedriver library, which enables the project to use Chrome as the default browser and apply a headless browser in order to run the code without any interface. Next, the BeautifulSoup library is used as the main parser for the website, where the library allows the code to parse all of the loaded HTML codes for the project to extract main keywords that match our pre-listed categories after the query is made by the user. After the final categorization is obtained for each word by repeating the process for each keyword, the project proceeds to the next stage of the process by passing the category back to the server to be sent to the machine learning model.

Multiprocessor systems have capabilities for parallel processing, which is advantageous in terms of the time it takes to run a computation, compared to serial computations that incur higher resources [25]. The ontology query's biggest issue is time. For the ontology to take each query one by one and crawl through each parent class, it is taking minutes, which on the user's end might be a serious inconvenience. In order to ensure the resources and time taken for the query given to the ontology are decreased, a multiprocessing library is used to query all the ingredients given by the user in a parallel manner. This method significantly decreases the time taken for the ontology to return the value, thus making the query much more efficient and user-friendly. Since the appendment of the result is done in this stage, Manager.list is used to ensuring a shared memory is used and no error arises after all results are returned.

Figure 13 below shows the result of the category received when a list of ingredients is sent to the ontology query.



Check Ingredient

```
GET /check/ham,milk,aspartame,apple,wine
```

```
HTTP 200 OK
Allow: GET, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "ingredient_given": [
        "ham",
        "milk",
        "aspartame",
        "apple",
        "wine"
    ],
    "result": [
        "pork",
        "dairy",
        "Chemical",
        "plant",
        "alcohol"
    ]
}
```

Figure 13. Results of multiple query by the server.

3.4 Machine Learning (ML) Classification

Machine Learning classification will be used in this project to classify which specific products belong to which interest group. From the various machine learning algorithms that can be applied in this project, several machine learning models were implemented to find the best performing model based on its metrics.

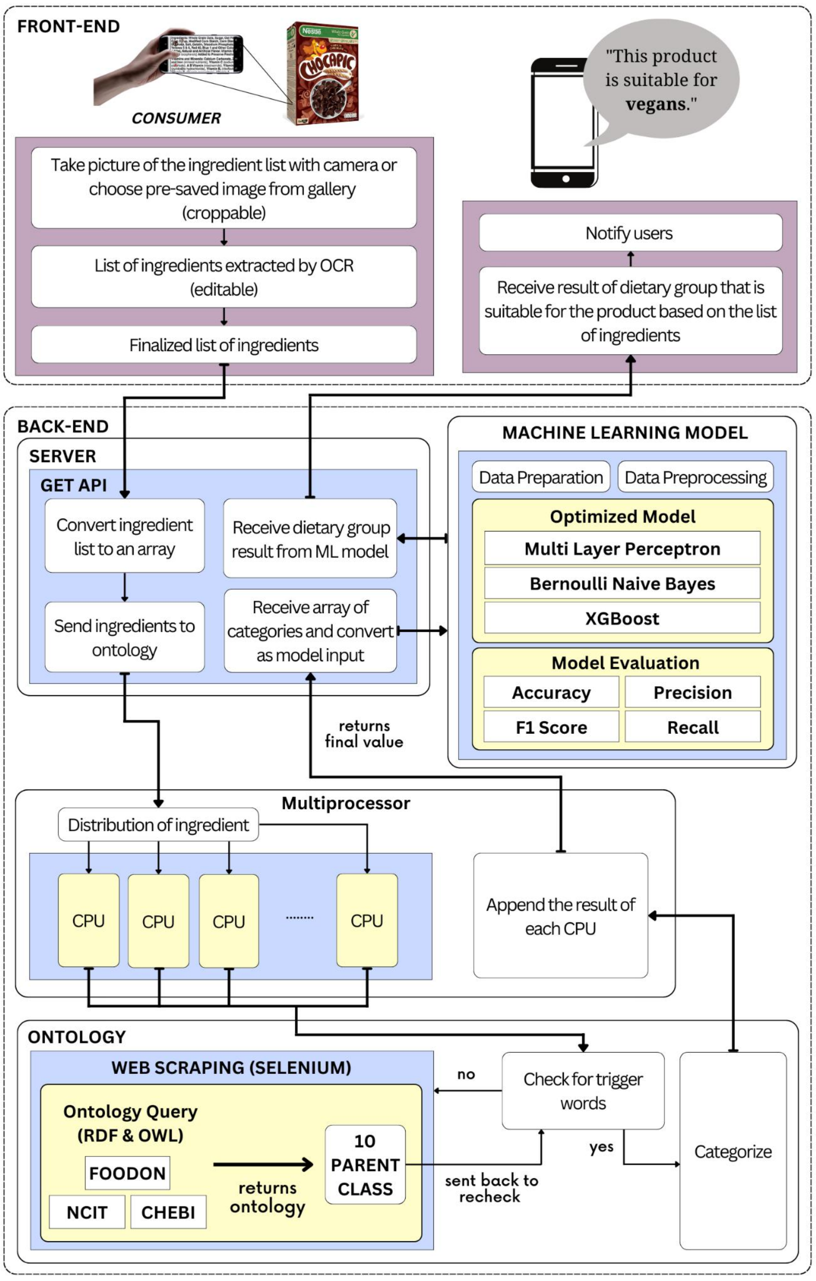
Beforehand, word embedding of the dataset needs to be performed in order to train the model. Afterwards, the model was trained through structured data classification. The process of selecting the best model began with the use of 'lazyclassifier' to train models with default parameters. From the 30 models provided in the 'lazyclassifier' library, 3 models were selected for optimization, and finally, from the selected and optimized models, the best performing model was selected for the prediction of the label. Further elaboration regarding the machine learning models that have been implemented can be found in section 4.4 below.

4. Implementation and Analysis Result

This section will explain how each part of the project is implemented in detail and how it synergizes with each other. This section will also analyse the results of the project and discuss further on the challenges and limitations that were faced throughout the progress of the project.

4.1 Implementation

Figure 14 below shows the overview and the framework of our project from the beginning until the end result is produced. The consumer will either take a picture of the ingredients of a product with their smartphone camera or use a preexisting image from their gallery. The application will extract the list of ingredients from the picture, which are editable. The finalized list of ingredients will then be sent to the back-end. The server will convert the ingredients into an array to be sent to the multiprocessor. The multiprocessor will then send the list to ontology, which will generate an output of ingredient categories. This output will be passed to the machine learning model, which will then produce the result of the dietary group. This result will be sent to the front-end again to notify the consumer whether the product is suitable for them or not.



FRONT-END
CONSUMER
"This product is suitable for vegans."
Take picture of the ingredient list with camera or choose pre-saved image from gallery (croppable)
List of ingredients extracted by OCR (editable)
Finalized list of ingredients
Notify users
Receive result of dietary group that is suitable for the product based on the list of ingredients
BACK-END
SERVER
GET API
Convert ingredient list to an array
Send ingredients to ontology
Receive dietary group result from ML model
Receive array of categories and convert as model input
MACHINE LEARNING MODEL
Data Preparation
Data Preprocessing
Optimized Model
Multi Layer Perceptron
Bernoulli Naive Bayes
XGBoost
Model Evaluation
Accuracy
Precision
F1 Score
Recall
returns final value
Multiprocessor
Distribution of ingredient
CPU
CPU
CPU
CPU
Append the result of each CPU
ONTOLOGY
WEB SCRAPING (SELENIUM)
Ontology Query (RDF & OWL)
FOODON
NCIT
CHEBI
returns ontology
10 PARENT CLASS
no
Check for trigger words
yes
sent back to recheck
Categorize

Figure 14. Project Framework.

4.1.1 Extraction of Information from Products

A mobile application is developed to provide a user interface and act as a scanner for the products. The scanner which is the device's camera is used to take the image of the product ingredient list which then will be extracted from the image by OCR before sending it to the backend. The extracted list of ingredients can be edited or deleted before the finalized ingredient list is sent to the server side to avoid any typos and increase the accuracy of the results.

4.1.2 Result Retrieval using API in Server

Django web framework is used for processing the information sent to the backend. GET API is used in the back-end to retrieve the result of the dietary preference. The server will first receive the list of ingredients from the front- end, which is then converted into an array. This array will be sent to the multiprocessor to be passed to the web scraping of the ontology. The server will receive the output of the category from the multiprocessor, which will then be sent to the machine learning model. The model will output the final result, which the server will receive and send to the front-end.

4.1.3 Multiprocessing & Ontology

Before sending the query of the ontology, the multiprocessing library is used to query each ingredient sent from the server simultaneously. This decreases the time taken for the query and ensures resources are used efficiently. On the ontology part, the ingredient is scanned through a list of trigger words. If it fits, the ingredient will be immediately categorized. If it does not, the word will go through ontology query, which is powered by the Selenium library. The library assures smooth ontology querying by the website and runs RDF and OWL code seamlessly, which in return will supply 10 parent classes of the ontology. The parent class will again be scanned through the trigger words, which will then be categorized. All the categories will then be appended using Manager.list, which ensures a shared memory, and sent back to the server.

4.1.4 Application of Machine Learning

The **data set** for this project is an ingredient list dataset that was found on Kaggle. This dataset contains information such as brand, categories, product, weight, product ingredients, and much more. For this project, only the product name and ingredients are required. Ingredients containing non-ASCII characters were removed beforehand. Then, several new columns are added to signify food categories, which are, poultry, beef, pork, dairy, egg, seafood, and grain. These categories make up for the diet groups that the project aims to help. **Table 2** below shows the categories and their relation to the food diet groups as previously mentioned:

Label	Vegan	Pescatarian	Pollotarian	Ovo Vegetarian	Lacto Vegetarian	Lacto-Ovo Vegetarian
Food Category Avoided	Animal Products	Meat	Beef, Pork	Meat, Seafood, Dairy	Meat, Seafood, Egg	Meat, Seafood

Table 2. Categories and Label for Food Diet Groups.

In this project, the ingredients from each product are extracted to create a smaller list of ingredients separated by commas. The smaller list of ingredients becomes an input for the ontology and the list of categories present in the product is returned as binary values for each category present in the product. Word embedding is performed on the categories and creates vectors. Finally, from the vectors, the model is trained and the labels are given through multi class classification. **Figure 15** shows the flow chart of the program.

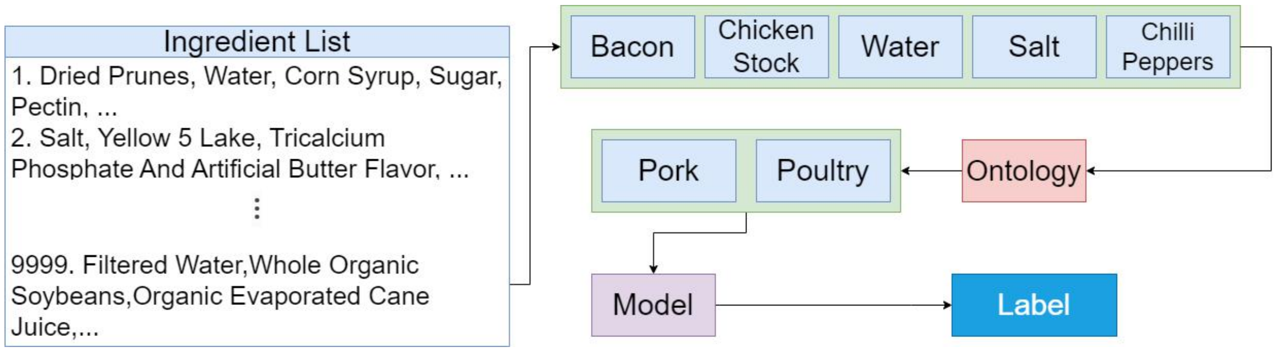
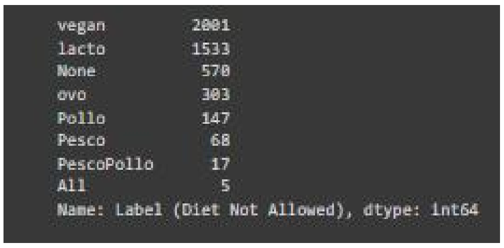

Ingredient List
1. Dried Prunes, Water, Corn Syrup, Sugar, Pectin, ...
2. Salt, Yellow 5 Lake, Tricalcium Phosphate And Artificial Butter Flavor, ...
⋮
9999. Filtered Water,Whole Organic Soybeans,Organic Evaporated Cane Juice,...
Bacon
Chicken Stock
Water
Salt
Chilli Peppers
Ontology
Pork
Poultry
Model
Label

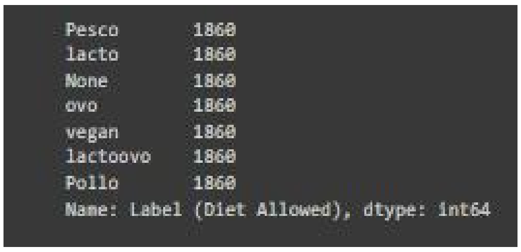
Figure 15. Ontology Approach.

In the beginning, there were 8 classes for labelling the diet preferences, consisting of vegan, lacto, none, ovo, pollo, pesco, pesco pollo, and all. Thus, if the diet preference label is vegan, vegans are not allowed to eat the product. However, the labels for each type of diet were imbalanced, and each of the products would need to be multi-labelled, creating an issue with the classification. As shown in **Figure 16**, labels 'all' and 'pesco pollo' have very little data compared to 'vegan' and 'lacto', so labels were inverted to indicate products that the users can consume. However, the pesco pollotarian diet still contained little amounts of data compared to the other labels. Therefore, the pesco pollotarian diet was removed, and the finalized classes were pesco, pollo, lacto, ovo, lacto ovo, none and vegan as shown in **Figure 17**.



```
vegan          2001
lacto          1533
None            570
ovo             303
Pollo           147
Pesco            68
PescoPollo       17
All               5
Name: Label (Diet Not Allowed), dtype: int64
```

Figure 16. Value Count of Labels before Balancing.



```
Pesco       1860
lacto       1860
None        1860
ovo         1860
vegan       1860
lactoovo    1860
Pollo       1860
Name: Label (Diet Allowed), dtype: int64
```

Figure 17. Value Count of Labels after Balancing.

Diet preferences of target groups become the labels for each classification. Since the labels are Objects, these labels were converted into numbers for data training. The number for each label are presented below as in **Table 3**:

Label	Vegan	Lacto	None	Ovo	Pollo	Pesco	Lacto-Ovo
Number	1	2	3	4	5	6	7

Table 3. Changing labels to numbers.

For **model training**, the dataset is split into 80:20 ratios for train and test data. In **Figure 18**, K-fold cross-validation is used on the dataset split to improve the randomness of the dataset in train and test. In this case, 5-fold cross-validation was applied, and five partitions of the full dataset were obtained. The partitions are evaluated through test performances, and the partition with the highest validation performance will be selected to train the model. Furthermore, by manipulating the value of the random state. Values from 1 to 10 are selected as the random state value, and K-Fold cross-validation is applied for each of the random states. This means that there are 50 different variations of datasets used for training and testing.

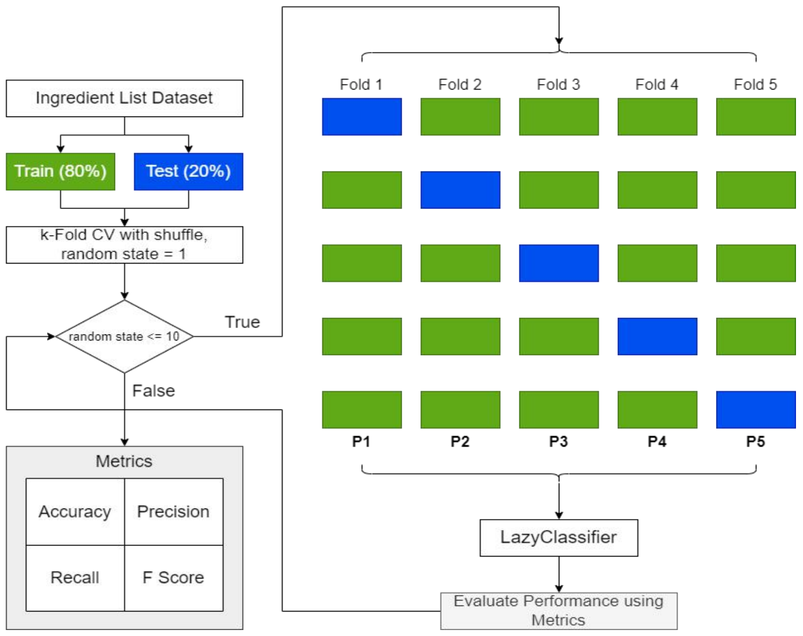
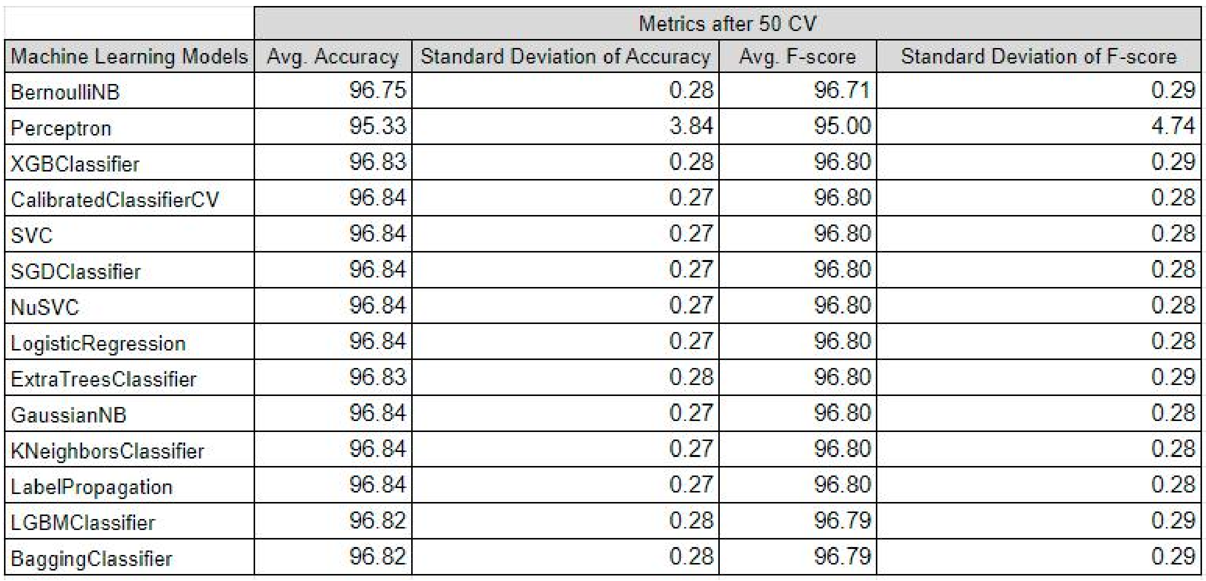


Figure 18. Data Preprocessing using K-Fold Cross-Validation.

Before selecting a specific model to train and optimize, 'lazyclassifier' was used to train 30 different models with default settings. In **Table 4**, most of the models resolved to around 95% to 96% accuracy and an f-score of 95 to 97. Furthermore, the standard deviation of each accuracy and f-score has a value of 0.28. From the table, the results of 15 out of the 30 models from the 'lazyclassifier' are presented. 3 models from different model families were selected as one of the possible final classification models: Bernoulli Naive Bayes, Multilayer Perceptron, and XGBoost Classifier.



	Metrics after 50 CV			
Machine Learning Models	Avg. Accuracy	Standard Deviation of Accuracy	Avg. F-score	Standard Deviation of F-score
BernoulliNB	96.75	0.28	96.71	0.29
Perceptron	95.33	3.84	95.00	4.74
XGBClassifier	96.83	0.28	96.80	0.29
CalibratedClassifierCV	96.84	0.27	96.80	0.28
SVC	96.84	0.27	96.80	0.28
SGDClassifier	96.84	0.27	96.80	0.28
NuSVC	96.84	0.27	96.80	0.28
LogisticRegression	96.84	0.27	96.80	0.28
ExtraTreesClassifier	96.83	0.28	96.80	0.29
GaussianNB	96.84	0.27	96.80	0.28
KNeighborsClassifier	96.84	0.27	96.80	0.28
LabelPropagation	96.84	0.27	96.80	0.28
LGBMClassifier	96.82	0.28	96.79	0.29
BaggingClassifier	96.82	0.28	96.79	0.29

Table 4. Accuracy and F-Score results from LazyClassifier.

Every machine learning model needs optimization to return the best results, thus hyperparameter tuning was used for each of the three models mentioned previously. Since the scikit-learn library provided the best customization of any model’s parameters, it was thoroughly explored for the optimization process. For the Bernoulli Naive Bayes model, the BernoulliNB( ) contains four customizable parameters, which are alpha, binarize, fit-prior, and class-prior. To elaborate, alpha refers to the additive smoothing parameter, binarize refers to the threshold for binarizing of sample features, fit_prior refers to determining the model’s ability in learning class prior probabilities, and class_prior refers to the prior probabilities of the classes [26].

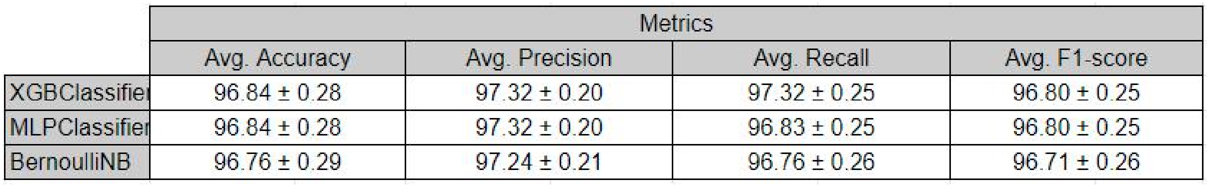
For the multilayer perceptron (MLP) model, there are more parameters for tuning. However, the parameters are affected by the weight optimizer function applied at every epoch. There are three different weight optimizer functions: the limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm(BFGS) or lbfgs in short, stochastic gradient descent (sgd) and adam, with the default optimizer being adam. Selecting either sgd or adam allows for more types of customization of the model’s hyperparameter tuning [26].

Lastly, for the XGBoost classifier, similar to the MLP model, there are many customizable parameter tuning possibilities. XGBoost uses similar concepts to a Decision Tree, thus most of its parameters are the same as a decision tree classifier. The parameters available for XGBoost are step size shrinkage (eta), minimum loss reduction (gamma), maximum depth of a tree (max_depth), the minimum sum of instance weight (min_child_weight), maximum delta step (max_delta_step), subsample ratio of the training instances (subsample), method of sampling training instances (sampling_method), subsampling, l1 and l2 regularization term on weights, control the balance of positive and negative weights, definition of tree updater sequence, types of boosting process, tree growth policy, maximum number of nodes, types of predictor algorithms, number of parallel trees, variable monotonicity constraints, and interaction constraints.

Then, the best parameter was selected by utilizing Randomized Search Cross Validation (RandomCV). Afterwards, similar to the process of testing the model’s performance as in the 'lazyclassifier', the optimized models are trained and tested on five folds and on 10 different random states resulting in 50 different variations of training and testing values. Next, the models are evaluated using metrics such as accuracy, precision, recall and f1-score.

4.2 Analysis Result

The table below shows the average value for each metrics evaluation.



	Metrics			
	Avg. Accuracy	Avg. Precision	Avg. Recall	Avg. F1-score
XGBClassifie	96.84 ± 0.28	97.32 ± 0.20	97.32 ± 0.25	96.80 ± 0.25
MLPClassifie	96.84 ± 0.28	97.32 ± 0.20	96.83 ± 0.25	96.80 ± 0.25
BernoulliNB	96.76 ± 0.29	97.24 ± 0.21	96.76 ± 0.26	96.71 ± 0.26

Table 5. Metrics after model optimization.

From the results of the optimized models in **Table 5**, the multilayer perceptron model was selected as the best performing model as its capabilities of increased metrics score after optimization of the models.

The performance of the application, the server and the ML Model seems to be seamless at the moment. However, in terms of ontology, it is too dependent on specific ontologies that we are utilizing for queries. Creating a new ontology database from scratch would cost too much time and there are too many factors to consider in food ontology, since we are not experts in food science. Thus, we decided that using available ontologies would be the best solution. In our ontology queries, 'trigger words' or keywords, were used, but, there are limitations in its capabilities in returning the food categories. Therefore, deeper research into the ontology would provide many other food categories that can be helpful in diversifying the target groups. Finally, an accuracy 96.8% was achieved by the optimized model, meaning that if the application was to be published, there is a possibility that it can be considered as a Minimum Viable Product (MVP).

4.3 Challenges and Limitations

Choosing Flutter as our main app development kit helped us a lot in creating the mobile app. Nevertheless, ensuring that the design, wanted features, and API aligns with the server was quite a challenge while developing the front end. As flutter is still new to the mobile development industry, searching for quality resources to be referred to takes more time than anticipated.

While Django and its REST Framework has been extremely helpful in the execution of this project, it does have some drawbacks. Django has predefined variables and files that are pertinent to the success of a Django app. Since it has its own ways to write the codes, and at a certain point there was a limitation as it is impossible to use our own file and code structures. It is important to take time to learn and understand the numerous rules of Django frameworks for deployment.

In the ontology part, deciding on obtaining the ontology from the party that made it or using Selenium to utilize web scraping while crawling through ontology is the biggest challenge that we faced. Having to learn a new language that is rarely used is a difficult task as the resources available online are limited. After failing to contact the party responsible for the ontology, web scraping was used and by doing so, we are limited to the code used by the Ontobee website. Having to study how the OWL and RDF are implemented in the website took quite some time and ensuring the efficiency and accuracy of the result took us 3 weeks. This stage has caused quite a delay in the project.

Multilayer Perceptrons allow many customizable features, however, optimizing the parameters for the model takes too much time. To elaborate, selecting either adam and sgd optimizer function allows for more customization on the other parameters, but these parameters are values in floats, therefore the possibilities are endless. On the other hand, even though Bernoulli Naive Bayes have a short execution time for the training, it only offers 4 parameters which constricts the amount of hyperparameter tuning that can be performed on the model. Other than that, doing research on XGBoost and ensuring the right parameter is used is quite a challenge. As the ML training is based on Google Colab, the free version comes with limited resources thus making it impossible to run GridSearchCV. After several tries, this is when RandomizedSearchCV is decided to be used as it takes fewer resources.

5. Conclusion and Impression

Dietary restrictions are not something that is to be under emphasized as there are many people in these groups that have dietary restrictions that are based on health, allergies, religious beliefs, or dietary preferences [27]. Nevertheless, it is inevitable for large food and beverage companies to produce food products that do not cater for all major and minority groups with specific dietary restrictions. Therefore, it is inconvenient for these people to shop for food products and find food freely and with ease due to their dietary restrictions. Our project can be used to push aside the uncertainty and help people in purchasing and consuming food products with confidence and less worry. Our experience from working on this project is that we realized how many food diet groups there are in this century and how hard it is for these people to search for suitable food products whether in their everyday life or while travelling. This pushes us more to make this project a success.

In this project, by scanning the ingredients list at the back of the packaging, the application will show the user each ingredient's origin, whether they contain specific chemicals, meat-based or plant-based ingredients, in addition to classifying if the product is consumable for particular groups or not in the end. The user will be able to take a picture or use a pre-existing image of ingredients from the gallery which then the app will extract it and send it to the back-end. The server will receive the list of ingredients via GET API and send them to the ontology query and classify through the machine learning model. The model will produce an output of the dietary preference, and the server will send this result back to the front-end. The app will notify the user whether the product is suitable with their dietary preference or not.

There are few challenges that we could not overcome that can be implemented for future improvements in this project. In addition to the dietary preferences that we have implemented, religious dietary restrictions can be included to provide assistance to wider diversity groups. Other than that, currently our application only accommodates products that are in English. So, a wider variety of language support can also be implemented. Finally, the efficiency of the ontology query in terms of time taken to return the result of the query can also be improved.

References

[1] A. Lang, “The 10 Most Common Dietary Restrictions,” *Healthline*, Jun. 07, 2021. https://www.healthline.com/nutrition/most-common-dietary-restrictions#1.-Lactose-intolerance (accessed Oct. 25, 2022).

[2] D. Wentworth, “What does it mean to be Halal certified?,” *Twin Rivers Technologies*, Apr. 09, 2018. http://www.twinriverstechnologies.com/blog/halal (accessed Oct. 24, 2022).

[3] A. Hayes, “Digital Nomad Definition,” *Investopedia*, Jul. 27, 2021. https://www.investopedia.com/terms/d/digital-nomad.asp (accessed Oct. 25, 2022).

[4] A. .-C. Boury-Brisset, “Ontology-based approach for information fusion,” in *Sixth International Conference of Information Fusion, 2003. Proceedings of the*, 2003, vol. 1, pp. 522–529. doi: 10.1109/ICIF.2003.177491.

[5] H.-J. Yoon, J.-C. O. Lee, K.-W. Park, and Y.-S. Hwang, “OCR 과 번역 API 를 활용한 외국인 대상 음식 정보 제공 웹앱 Web-app that provides food information for foreigners using OCR and translation API”, Accessed: Oct. 25, 2022. [Online]. Available: https://developers.kakao.com/doc

[6] C.-H. Chen, M. Karvela, M. Sohbati, T. Shinawatra, and C. Toumazou, “PERSON—Personalized Expert Recommendation System for Optimized Nutrition,” *IEEE Trans Biomed Circuits Syst*, vol. 12, no. 1, pp. 151–160, 2018, doi: 10.1109/TBCAS.2017.2760504.

[7] A. F. Coskun, J. Wong, D. Khodadadi, R. Nagi, A. Tey, and A. Ozcan, “A personalized food allergen testing platform on a cellphone,” *Lab Chip*, vol. 13, no. 4, pp. 636–640, 2013, doi: 10.1039/C2LC41152K.

[8] D. Celik Ertuğrul, “FoodWiki: a Mobile App Examines Side Effects of Food Additives Via Semantic Web,” *J Med Syst*, vol. 40, Nov. 2015, doi: 10.1007/s10916-015-0372-6.

[9] S. Abdullah *et al.*, “Food Code Breaker (FCB)-Developing the Multi-Lingual Food Code Translation Android Application Using Multi-Options Code Reader System,” p. 12013, 2018, doi: 10.1088/1742-6596/1019/1/012013.

[10] S. Prajapati, “Eatable: An Application That Helps People with Food Allergies Check and Locate Allergen-Free Food Products,” 2017.

[11] S. Junaini and J. Abdullah, *MyMobiHalal 2.0: Malaysian mobile Halal product verification using camera phone barcode scanning and MMS*. 2008. doi: 10.1109/ICCCE.2008.4580659.

[12] R. P. Abao, C. v Malabanan, and A. P. Galido, “Design and Development of FoodGo: A Mobile Application using Situated Analytics to Augment Product Information,” *Procedia Comput Sci*, vol. 135, pp. 186–193, 2018, doi: https://doi.org/10.1016/j.procs.2018.08.165.

[13] E. Dunford *et al.*, “FoodSwitch: A Mobile Phone App to Enable Consumers to Make Healthier Food Choices and Crowdsourcing of National Food Composition Data,” *JMIR mHealth uHealth*, vol. 2, no. 3, p. e37, Aug. 2014, [Online]. Available: http://mhealth.jmir.org/2014/3/e37/

[14] A. A. Shari *et al.*, “Mobile Application of Food Recommendation For Allergy Baby Using Rule-Based Technique,” in *2019 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS)*, 2019, pp. 273–278. doi: 10.1109/I2CACIS.2019.8825026.

[15] N. Kuzmin, K. Ignatiev, and D. Grafov, “Experience of Developing a Mobile Application Using Flutter,” *Lecture Notes in Electrical Engineering*, vol. 621, pp. 571–575, 2020, doi: 10.1007/978-981-15-1465-4_56.

[16] W. Zabierowski and K. Wasilewski, “A Comparison of Java, Flutter and Kotlin/Native Technologies for Sensor Data-Driven Applications,” *Sensors*, vol. 21, May 2021, doi: 10.3390/S21103324.

[17]	Z. Mosciski, “What Is SHA-256 Algorithm: How it Works and Applications [2022 Edition] | Simplilearn (2022).”https://excoga.ngontinh24.com/article/what-is-sha-256-algorithm-how-it-works-and-applications-2022-edition-simplilearn (accessed Oct. 26, 2022).

[18]	Pub Developer, “simple_ocr_plugin | Flutter Package,” *pub.dev*, 2020. https://pub.dev/packages/simple_ocr_plugin (accessed Oct. 26, 2022).

[19]	Data Flair Team, “Django Advantages and Disadvantages - Why You Should Choose Django? - DataFlair,” *Data Flair*. https://data-flair.training/blogs/django-advantages-and-disadvantages/ (accessed Oct. 26, 2022).

[20]	D. Memberives, “What is Django Rest Framework and why you should learn it - Let’s learn about,” *Let’s learn about*, Aug. 20, 2019. https://letslearnabout.net/blog/what-is-django-rest-framework-and-why-you-should-learn-it/ (accessed Oct. 26, 2022).

[21]	B. Kumar, “Difference Between App And Project In Django - Python Guides,” *PythonGuides.com*, Aug. 12, 2021. https://pythonguides.com/django-app-vs-project/ (accessed Oct. 26, 2022).

[22]	maximillian.pippi, “PythonAnywhere: un ambiente Python nel proprio browser | HTML.it,” *HTML.IT*, May 10, 2012. https://www.html.it/magazine/pythonanywhere-un-ambiente-python-nel-proprio-browser/ (accessed Oct. 26, 2022)

[23]	J. Wu, I. Ilyas, and G. Weddell, “A Study of Ontology-based Query Expansion”.

[24]	J. Bhogal, A. Macfarlane, and P. Smith, “A review of ontology based query expansion,” *Inf Process Manag*, vol. 43, no. 4, pp. 866–886, 2007, doi: https://doi.org/10.1016/j.ipm.2006.09.003.

[25]	E.A. Trakhtengerts, Yu. M. Shuraits,"Optimization of Computations on Multiprocessor Systems", IFAC Proceedings Volumes, Volume 15, Issue 7, 1982, Pages 89-93, doi: https://doi.org/10.1016/S1474-6670(17)62805-2.

[26]	F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

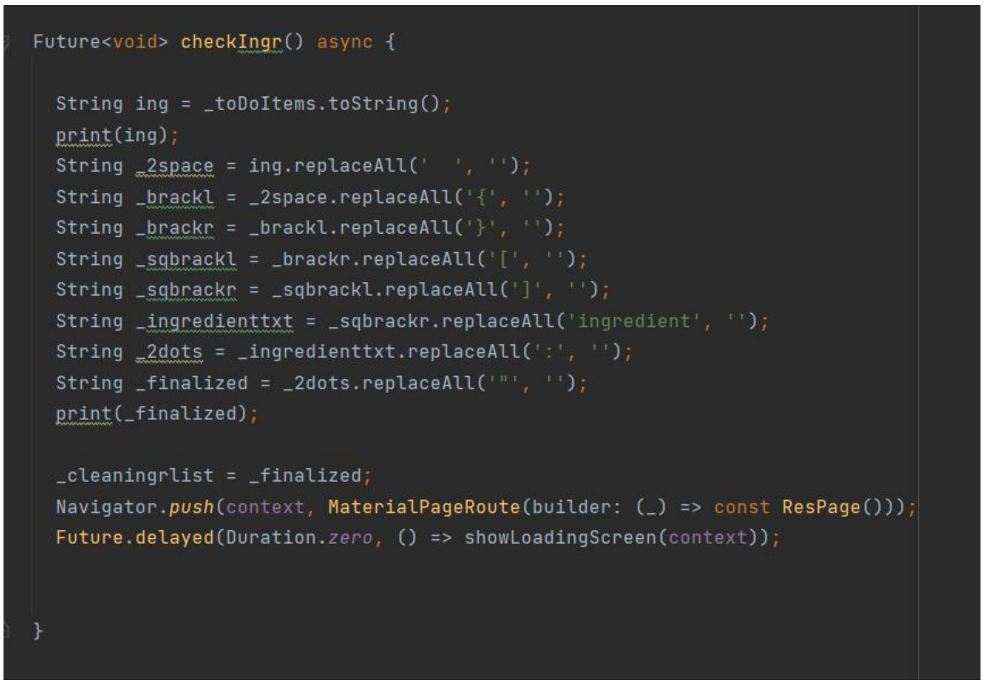
[27]	C. Price, “What Are the Most Common Dietary Restrictions? - On the Line | Toast POS,” *On the Line*. https://pos.toasttab.com/blog/on-the-line/common-dietary-restrictions (accessed Oct. 26, 2022).

APPENDIX

In this section, we attached screenshots of codes from the project that exhibit the important parts for each section of the project. The order of the codes are Front-end, Server, Ontology and Machine Learning.

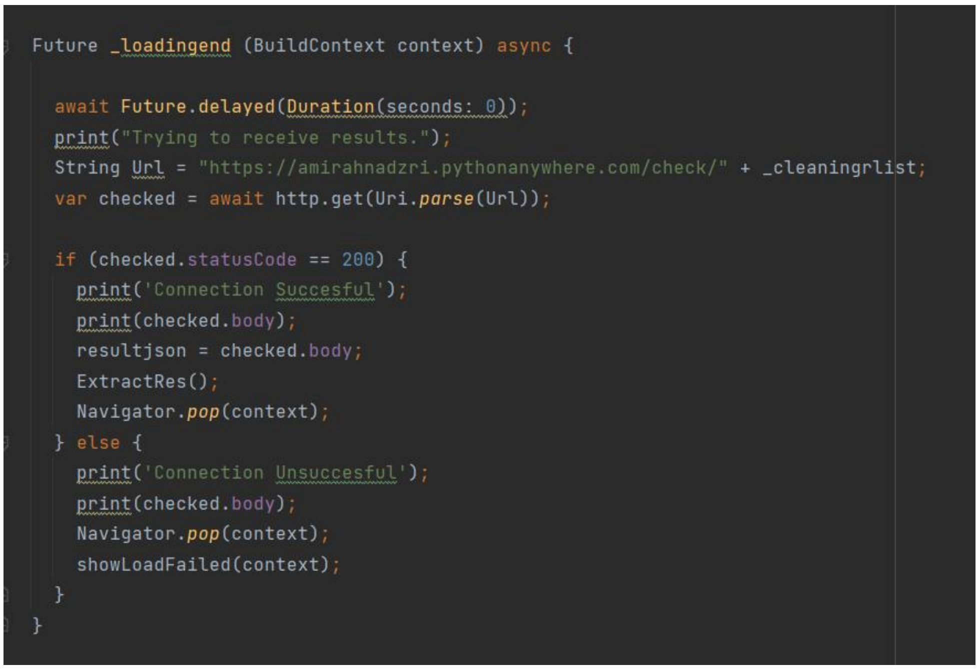
The full code for this project can be found through this link: https://github.com/amirahnadzri/fypsoftware

1) Front-end:



```
Future<void> checkIngr() async {

  String ing = _toDoItems.toString();
  print(ing);
  String _2space = ing.replaceAll('  ', '');
  String _brackl = _2space.replaceAll('{', '');
  String _brackr = _brackl.replaceAll('}', '');
  String _sqbrackl = _brackr.replaceAll('[', '');
  String _sqbrackr = _sqbrackl.replaceAll(']', '');
  String _ingredienttxt = _sqbrackr.replaceAll('ingredient', '');
  String _2dots = _ingredienttxt.replaceAll(':', '');
  String _finalized = _2dots.replaceAll('"', '');
  print(_finalized);

  _cleaningrlist = _finalized;
  Navigator.push(context, MaterialPageRoute(builder: (_) => const ResPage()));
  Future.delayed(Duration.zero, () => showLoadingScreen(context));


}
```

Code 1. checkIngr() function to preprocess data before passing to server.

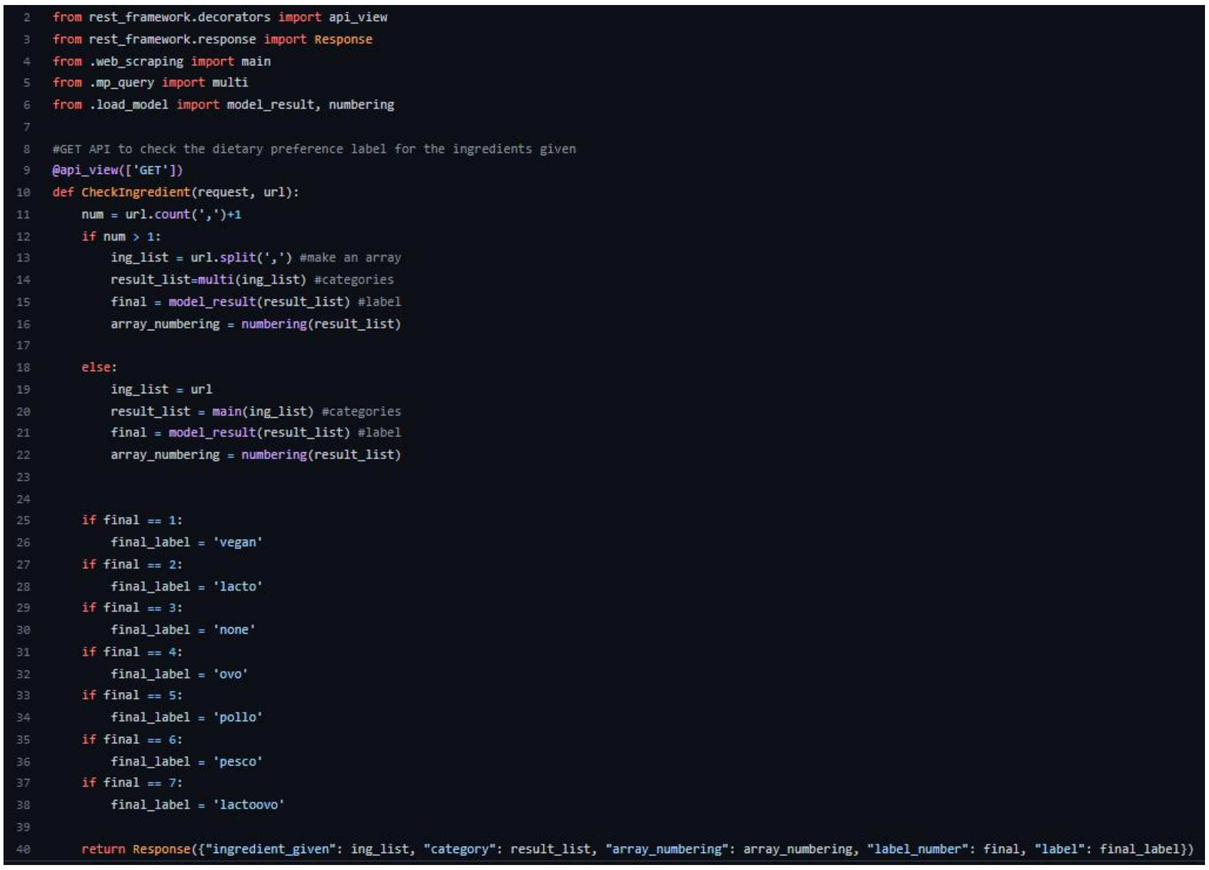


```
Future _loadingend (BuildContext context) async {

  await Future.delayed(Duration(seconds: 0));
  print("Trying to receive results.");
  String Url = "https://amirahnadzri.pythonanywhere.com/check/" + _cleaningrlist;
  var checked = await http.get(Uri.parse(Url));

  if (checked.statusCode == 200) {
    print('Connection Succesful');
    print(checked.body);
    resultjson = checked.body;
    ExtractRes();
    Navigator.pop(context);
  } else {
    print('Connection Unsuccesful');
    print(checked.body);
    Navigator.pop(context);
    showLoadFailed(context);
  }
}
```

Code 2. Sending finalized list of ingredients to the server API.

The main feature of the app is checking of ingredients by sending the list of ingredients to the server side. Therefore, in Code 1, checkIngr() is the function that will run when the ‘check’ button is pressed. The string of ingredients undergoes data preprocessing to ensure the string is free from any unwanted characters or spaces. The finalized string are then sent to the API server by adding the string to the end of the server API url to retrieve results on the dietary group sent back by the server as json body as depicted in Code 2.

2) Server:



```python
from rest_framework.decorators import api_view
from rest_framework.response import Response
from .web_scraping import main
from .mp_query import multi
from .load_model import model_result, numbering

#GET API to check the dietary preference label for the ingredients given
@api_view(['GET'])
def CheckIngredient(request, url):
    num = url.count(',')+1
    if num > 1:
        ing_list = url.split(',') #make an array
        result_list=multi(ing_list) #categories
        final = model_result(result_list) #label
        array_numbering = numbering(result_list)

    else:
        ing_list = url
        result_list = main(ing_list) #categories
        final = model_result(result_list) #label
        array_numbering = numbering(result_list)


    if final == 1:
        final_label = 'vegan'
    if final == 2:
        final_label = 'lacto'
    if final == 3:
        final_label = 'none'
    if final == 4:
        final_label = 'ovo'
    if final == 5:
        final_label = 'pollo'
    if final == 6:
        final_label = 'pesco'
    if final == 7:
        final_label = 'lactoovo'

    return Response({"ingredient_given": ing_list, "category": result_list, "array_numbering": array_numbering, "label_number": final, "label": final_label})
```

Code 3. GET API to retrieve result of dietary preference.

Code 3 shows the GET API that is used to receive ingredients from the front-end, which then is converted to an array. The array will be sent to the ontology for querying and machine learning. The output from machine learning is included in the return Response, which is what the front-end will receive.

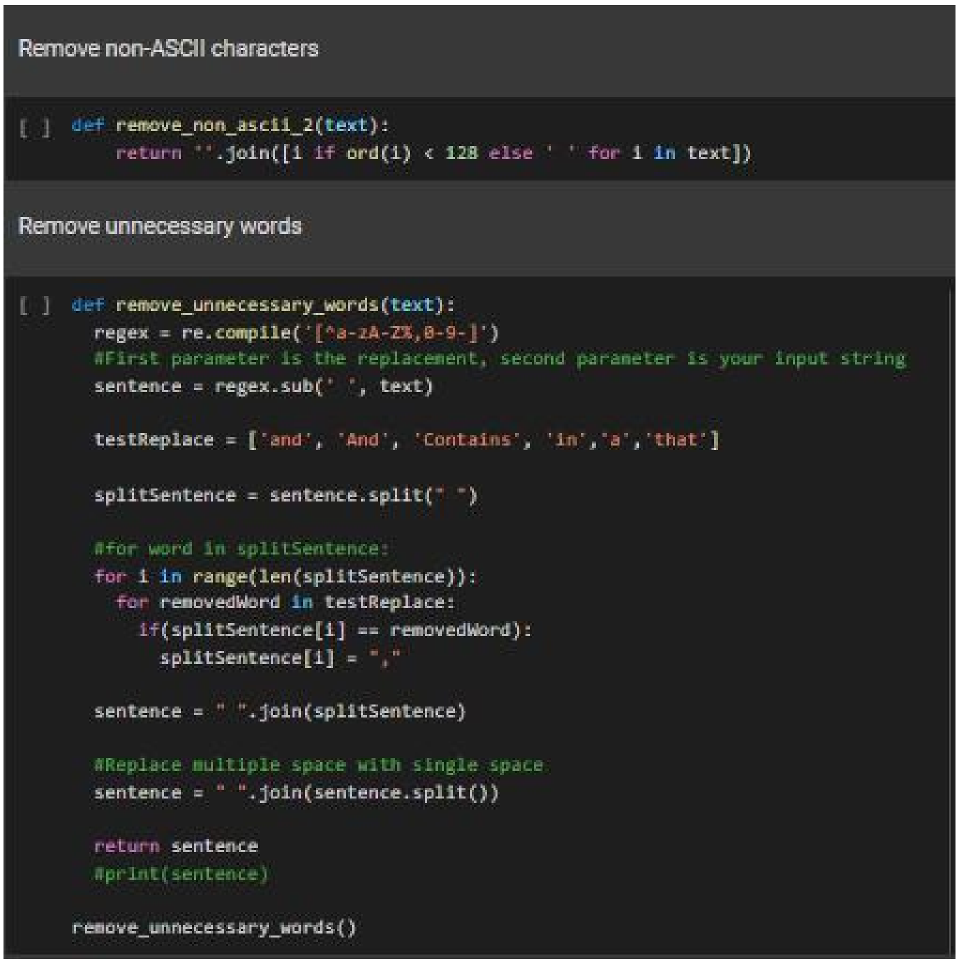
3) Ontology:

```python
def main(ingredient):
  test = ingredient.split(" ")
  ingredient = ingredient.replace(" ","+")
  trigger = [ "corn", "seed", "beans", "peas", "soy", "wine","coconut","cream","rice"]
  test = [each_string.lower() for each_string in test]
  if final_category(test) == "not in category" or final_category(test) == "nut":
    count2 = 0
    for find in trigger:
      str_match = list(filter(lambda x: find in x, test))
      if str_match:
        break
      count2 = count2 + 1
      if count2 == 9:
        break
    if count2 < 9:
      if (trigger[count2] == "corn") or (trigger[count2] == "coconut") or (trigger[count2] == "seed") or (trigger[count2] == "corn") or
        (trigger[count2] == "peas") or (trigger[count2] == "beans") or (trigger[count2] == "soy"):
        return "plant"
      elif (trigger[count2] == "cream"):
        return "milk"
      elif (trigger[count2] == "wine"):
        return "alcohol"
      elif (trigger[count2] == "rice"):
        return "grain"
  try:
    keyword_id = get_idfromkeyword(ingredient)
  except IndexError:
    ingredient = ingredient.split("+")
    if len(ingredient) == 0:
      return "not in ontology"
    else:
      del ingredient[0]
      ingredient = "+".join(ingredient)
      return main(ingredient)
  chebi_check = keyword_id[0:5]
  keyword_id2 = get_idfromkeyword_chebicheck(ingredient)
  chebi_check2 = keyword_id2[0:5]
  if chebi_check == "CHEBI" :
    return "Chemical"
  elif chebi_check2 == "CHEBI" :
    return "Chemical"
  else:
    foodon_check = keyword_id[0:6]
    if not foodon_check == "FOODON":
      keyword_id = get_idfromkeyword2(ingredient)
    category_list = get_category(keyword_id, ingredient,i)
    result = final_category(category_list)
    if result == "alcohol":
      category_list = get_category_alcohol(keyword_id, ingredient,i)
      result = final_category(category_list)

  return result
```

Code 4. Ontology Query main function.

Code 4 shows the ontology query that utilizes the Selenium Library to receive ingredients from the back-end. This code is run simultaneously using the multiprocessing library. The output, which is the category of the ingredient, will be appended by the multiprocessing library that will then be returned to the back-end.

4) Machine Learning:



Remove non-ASCII characters

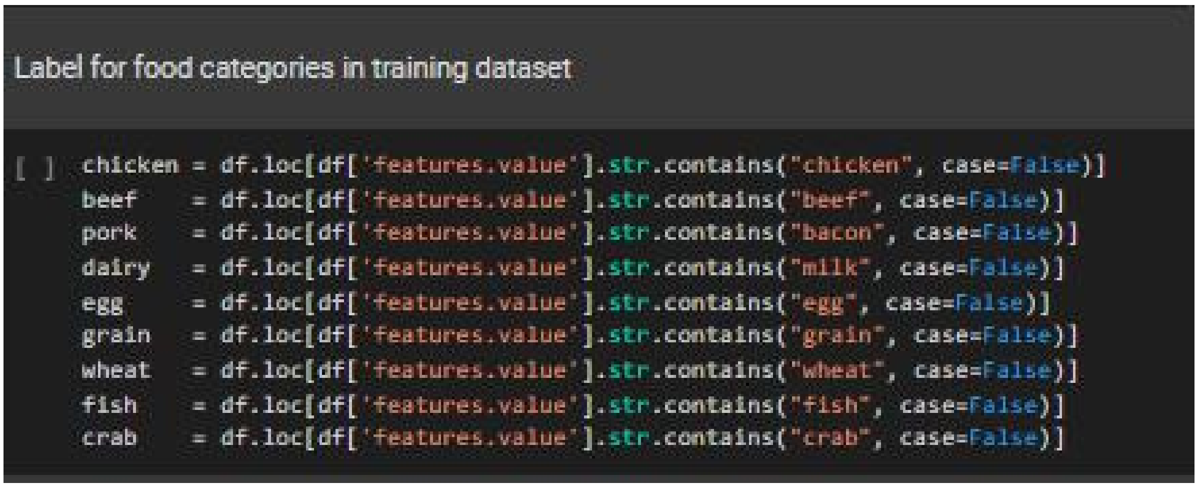
```
[ ] def remove_non_ascii_2(text):
        return ''.join([i if ord(i) < 128 else ' ' for i in text])
```

Remove unnecessary words

```
[ ] def remove_unnecessary_words(text):
      regex = re.compile('[^a-zA-Z%,0-9-]')
      #First parameter is the replacement, second parameter is your input string
      sentence = regex.sub(' ', text)

      testReplace = ['and', 'And', 'Contains', 'in','a','that']

      splitSentence = sentence.split(" ")

      #for word in splitSentence:
      for i in range(len(splitSentence)):
        for removedWord in testReplace:
          if(splitSentence[i] == removedWord):
            splitSentence[i] = ","

      sentence = " ".join(splitSentence)

      #Replace multiple space with single space
      sentence = " ".join(sentence.split())

      return sentence
      #print(sentence)

    remove_unnecessary_words()
```

Code 5. Remove non-ASCII characters and null values.

In Code 5, these functions were used to remove non-ASCII characters and remove unnecessary words such as ‘contains, ‘and, ‘in’, ‘that’, etc. This allows improved readability of the product ingredients for the ontology query in order to label the categories of food present in the product.

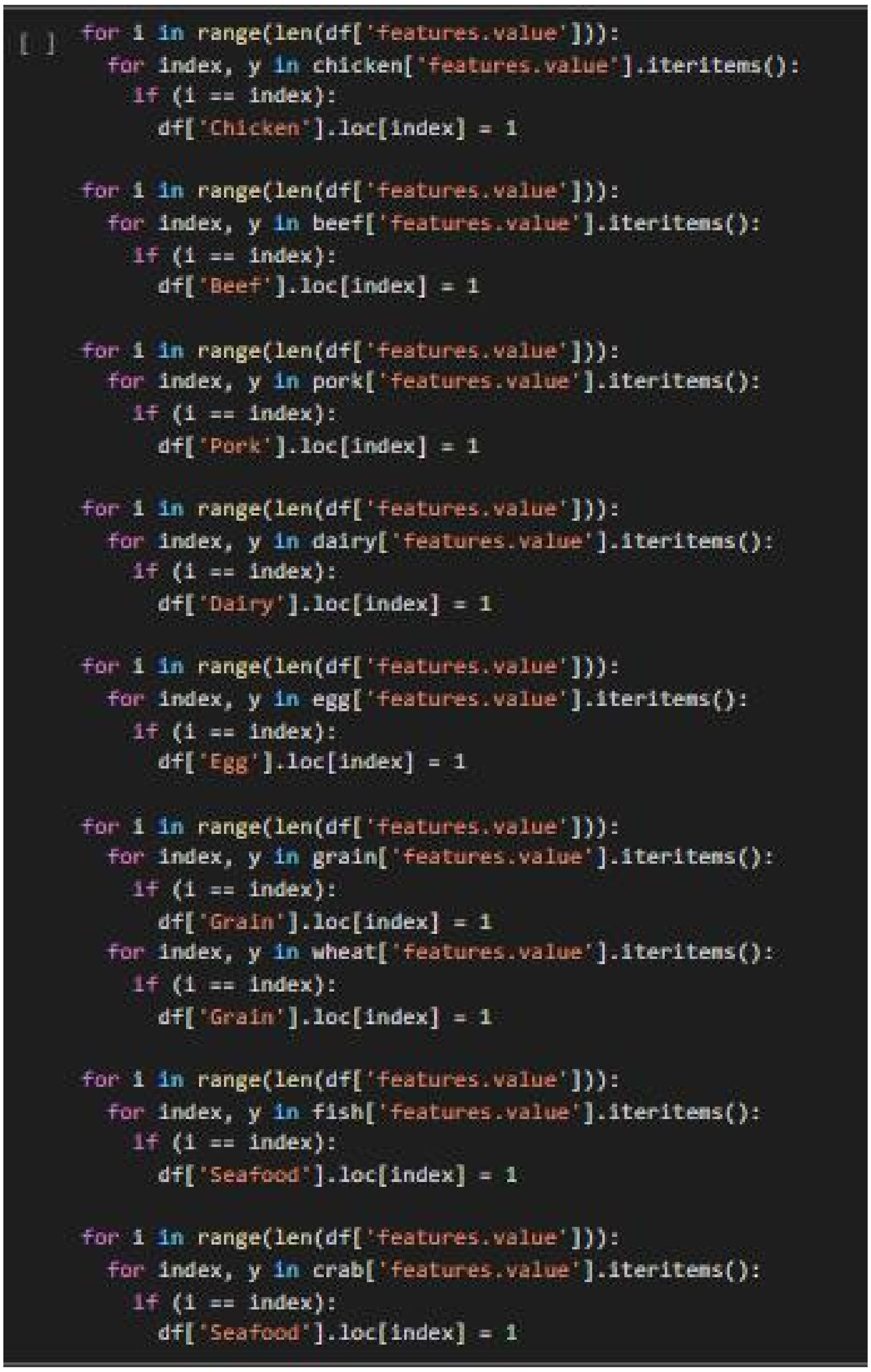


Label for food categories in training dataset

```
[ ] chicken = df.loc[df['features.value'].str.contains("chicken", case=False)]
    beef    = df.loc[df['features.value'].str.contains("beef", case=False)]
    pork    = df.loc[df['features.value'].str.contains("bacon", case=False)]
    dairy   = df.loc[df['features.value'].str.contains("milk", case=False)]
    egg     = df.loc[df['features.value'].str.contains("egg", case=False)]
    grain   = df.loc[df['features.value'].str.contains("grain", case=False)]
    wheat   = df.loc[df['features.value'].str.contains("wheat", case=False)]
    fish    = df.loc[df['features.value'].str.contains("fish", case=False)]
    crab    = df.loc[df['features.value'].str.contains("crab", case=False)]
```

Code 6. Extracting list of products containing each food category.

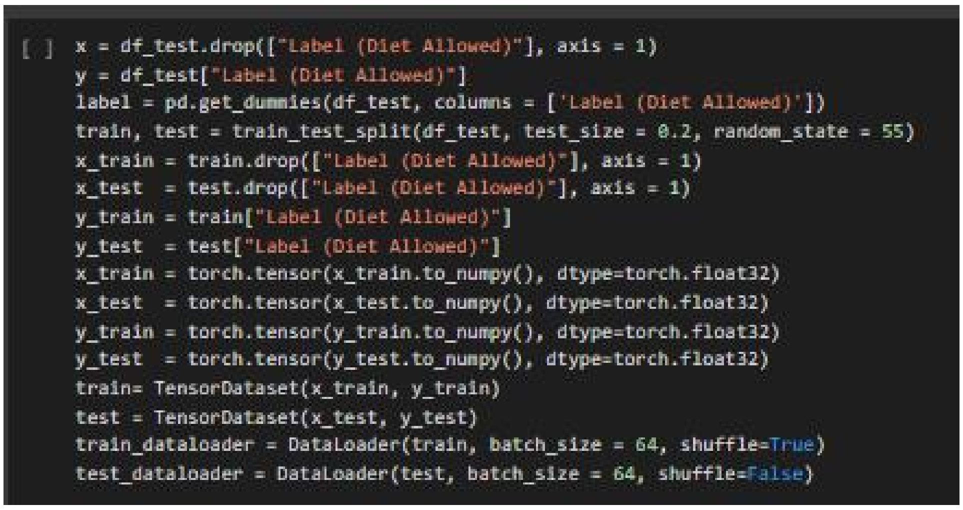
In Code 6, the code for creating a list of food products that contains certain ingredients that will be used to label the food category columns in the dataset.



```
[ ]  for i in range(len(df['features.value'])):
       for index, y in chicken['features.value'].iteritems():
         if (i == index):
           df['Chicken'].loc[index] = 1

     for i in range(len(df['features.value'])):
       for index, y in beef['features.value'].iteritems():
         if (i == index):
           df['Beef'].loc[index] = 1

     for i in range(len(df['features.value'])):
       for index, y in pork['features.value'].iteritems():
         if (i == index):
           df['Pork'].loc[index] = 1

     for i in range(len(df['features.value'])):
       for index, y in dairy['features.value'].iteritems():
         if (i == index):
           df['Dairy'].loc[index] = 1

     for i in range(len(df['features.value'])):
       for index, y in egg['features.value'].iteritems():
         if (i == index):
           df['Egg'].loc[index] = 1

     for i in range(len(df['features.value'])):
       for index, y in grain['features.value'].iteritems():
         if (i == index):
           df['Grain'].loc[index] = 1
       for index, y in wheat['features.value'].iteritems():
         if (i == index):
           df['Grain'].loc[index] = 1

     for i in range(len(df['features.value'])):
       for index, y in fish['features.value'].iteritems():
         if (i == index):
           df['Seafood'].loc[index] = 1

     for i in range(len(df['features.value'])):
       for index, y in crab['features.value'].iteritems():
         if (i == index):
           df['Seafood'].loc[index] = 1
```

Code 7. Binarized food category columns.

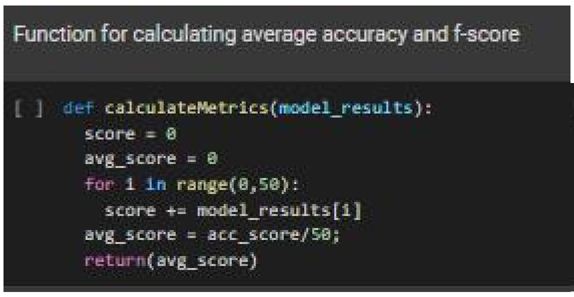
By utilizing the previous list of ingredients, the columns are labelled ‘1’ on indexes that contain the specific food category in its ingredients. Then, categories with label ‘1’ were used to label each product based on a specific target group as seen in Code 7.



```
x = df_test.drop(["Label (Diet Allowed)"], axis = 1)
y = df_test["Label (Diet Allowed)"]
label = pd.get_dummies(df_test, columns = ['Label (Diet Allowed)'])
train, test = train_test_split(df_test, test_size = 0.2, random_state = 55)
x_train = train.drop(["Label (Diet Allowed)"], axis = 1)
x_test  = test.drop(["Label (Diet Allowed)"], axis = 1)
y_train = train["Label (Diet Allowed)"]
y_test  = test["Label (Diet Allowed)"]
x_train = torch.tensor(x_train.to_numpy(), dtype=torch.float32)
x_test  = torch.tensor(x_test.to_numpy(), dtype=torch.float32)
y_train = torch.tensor(y_train.to_numpy(), dtype=torch.float32)
y_test  = torch.tensor(y_test.to_numpy(), dtype=torch.float32)
train= TensorDataset(x_train, y_train)
test = TensorDataset(x_test, y_test)
train_dataloader = DataLoader(train, batch_size = 64, shuffle=True)
test_dataloader = DataLoader(test, batch_size = 64, shuffle=False)
```

Code 8. Data Preprocessing.

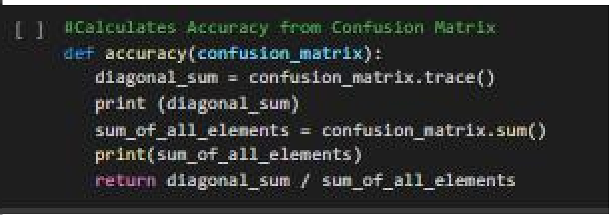
As seen in Code 8, after labelling the food products, the labels are then converted into numbers for data preprocessing.



Function for calculating average accuracy and f-score

```
def calculateMetrics(model_results):
  score = 0
  avg_score = 0
  for i in range(0,50):
    score += model_results[i]
  avg_score = acc_score/50;
  return(avg_score)
```

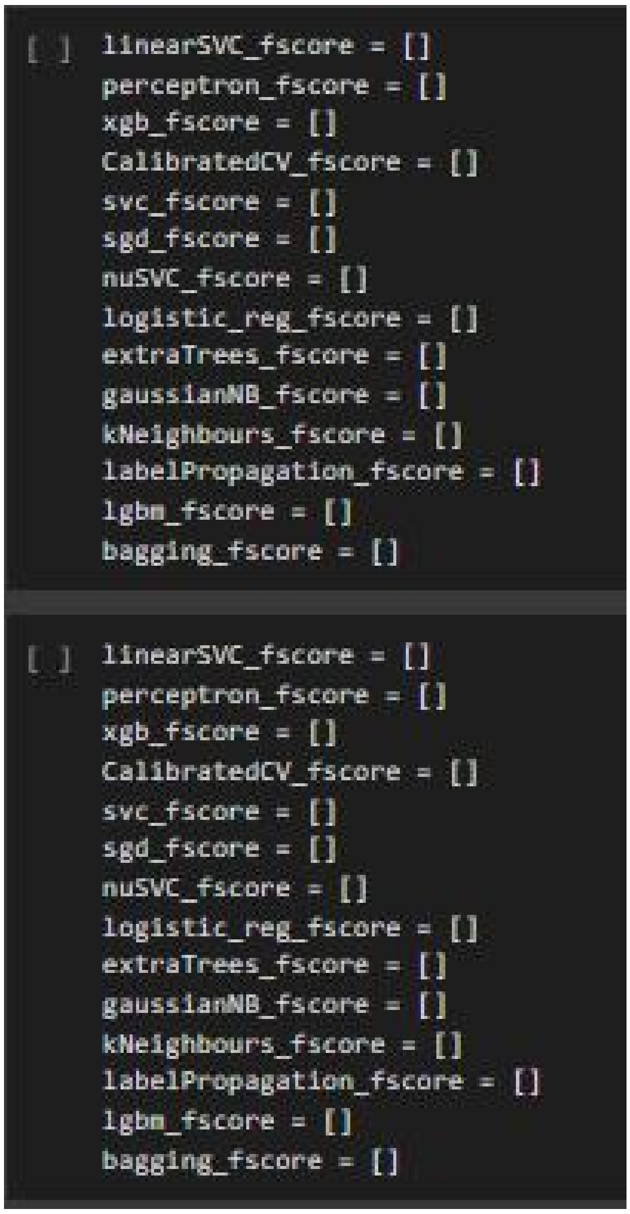
Code 9. Function for calculating average accuracy and f-score.



```
#Calculates Accuracy from Confusion Matrix
def accuracy(confusion_matrix):
   diagonal_sum = confusion_matrix.trace()
   print (diagonal_sum)
   sum_of_all_elements = confusion_matrix.sum()
   print(sum_of_all_elements)
   return diagonal_sum / sum_of_all_elements
```

Code 10. Function for calculating accuracy.

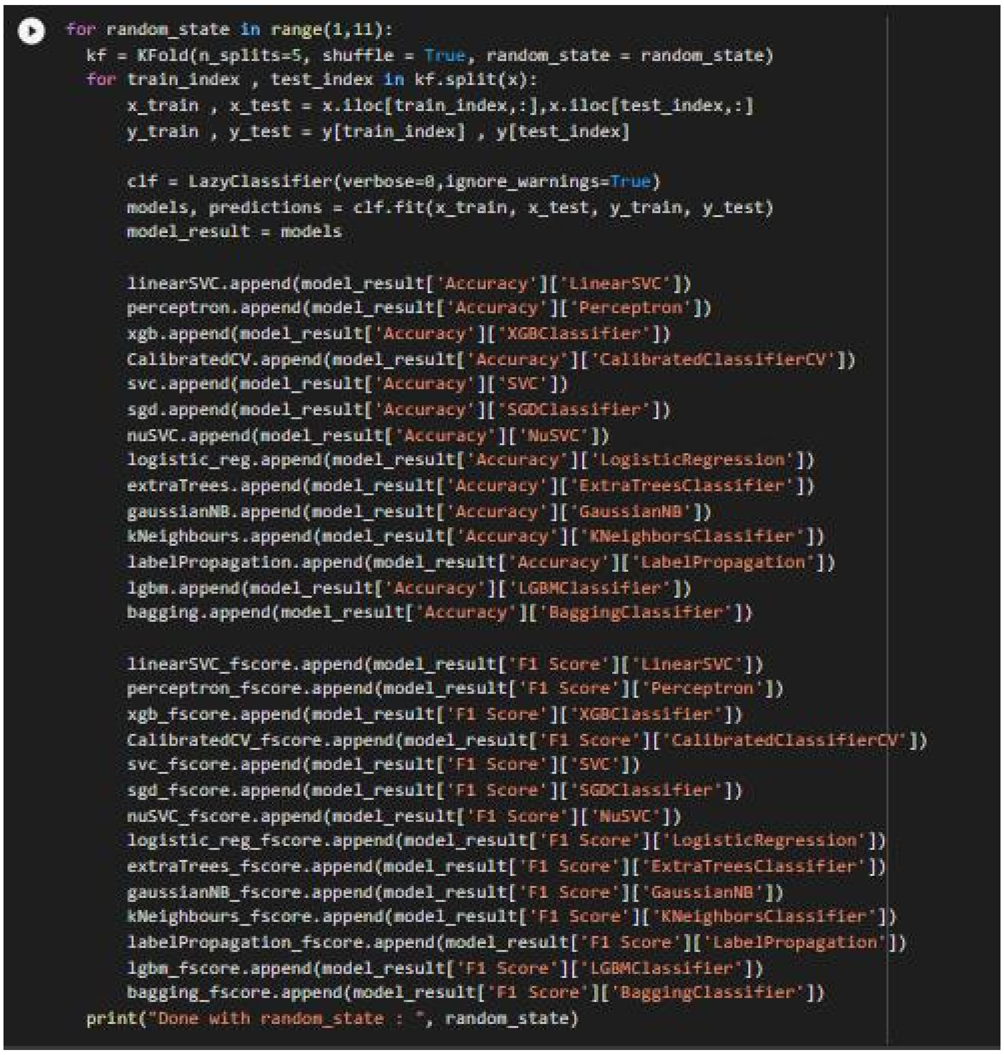
These functions in Code 9 and Code 10 were used in calculating the accuracy and f-score during model evaluations. The calculateMetrics function was used in evaluating the metrics in the lazyclassifier models and the accuracy function was used to evaluate the three optimized models.


```
[ ]  linearSVC_fscore = []
     perceptron_fscore = []
     xgb_fscore = []
     CalibratedCV_fscore = []
     svc_fscore = []
     sgd_fscore = []
     nuSVC_fscore = []
     logistic_reg_fscore = []
     extraTrees_fscore = []
     gaussianNB_fscore = []
     kNeighbours_fscore = []
     labelPropagation_fscore = []
     lgbm_fscore = []
     bagging_fscore = []
```

```
[ ]  linearSVC_fscore = []
     perceptron_fscore = []
     xgb_fscore = []
     CalibratedCV_fscore = []
     svc_fscore = []
     sgd_fscore = []
     nuSVC_fscore = []
     logistic_reg_fscore = []
     extraTrees_fscore = []
     gaussianNB_fscore = []
     kNeighbours_fscore = []
     labelPropagation_fscore = []
     lgbm_fscore = []
     bagging_fscore = []
```

Code 11. List of values for each model in LazyClassifier.

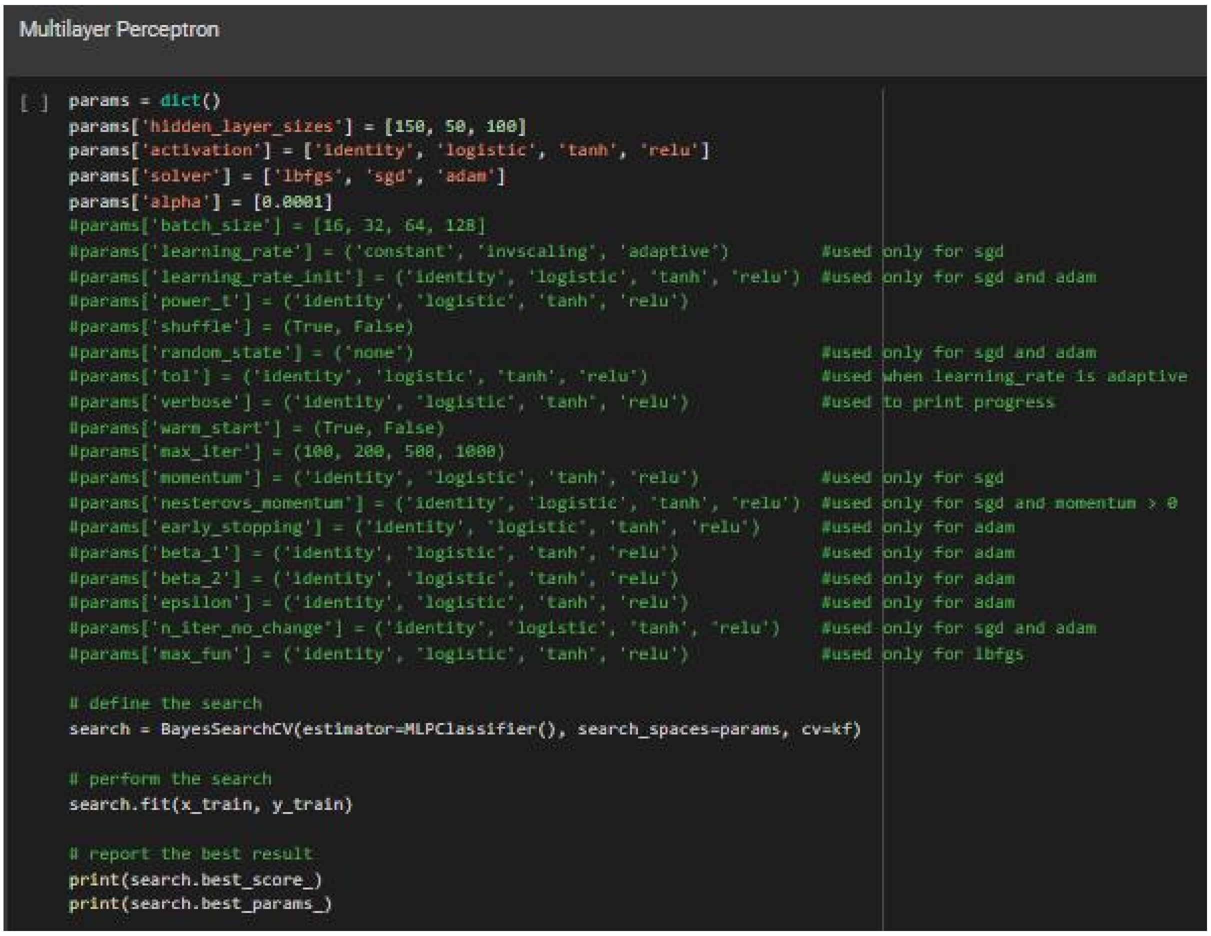
These codes in Code 11 are the lists created to store results of metrics evaluation for each loop in the ‘lazyclassifier’.



```
for random_state in range(1,11):
  kf = KFold(n_splits=5, shuffle = True, random_state = random_state)
  for train_index , test_index in kf.split(x):
      x_train , x_test = x.iloc[train_index,:],x.iloc[test_index,:]
      y_train , y_test = y[train_index] , y[test_index]

      clf = LazyClassifier(verbose=0,ignore_warnings=True)
      models, predictions = clf.fit(x_train, x_test, y_train, y_test)
      model_result = models

      linearSVC.append(model_result['Accuracy']['LinearSVC'])
      perceptron.append(model_result['Accuracy']['Perceptron'])
      xgb.append(model_result['Accuracy']['XGBClassifier'])
      CalibratedCV.append(model_result['Accuracy']['CalibratedClassifierCV'])
      svc.append(model_result['Accuracy']['SVC'])
      sgd.append(model_result['Accuracy']['SGDClassifier'])
      nuSVC.append(model_result['Accuracy']['NuSVC'])
      logistic_reg.append(model_result['Accuracy']['LogisticRegression'])
      extraTrees.append(model_result['Accuracy']['ExtraTreesClassifier'])
      gaussianNB.append(model_result['Accuracy']['GaussianNB'])
      kNeighbours.append(model_result['Accuracy']['KNeighborsClassifier'])
      labelPropagation.append(model_result['Accuracy']['LabelPropagation'])
      lgbm.append(model_result['Accuracy']['LGBMClassifier'])
      bagging.append(model_result['Accuracy']['BaggingClassifier'])

      linearSVC_fscore.append(model_result['F1 Score']['LinearSVC'])
      perceptron_fscore.append(model_result['F1 Score']['Perceptron'])
      xgb_fscore.append(model_result['F1 Score']['XGBClassifier'])
      CalibratedCV_fscore.append(model_result['F1 Score']['CalibratedClassifierCV'])
      svc_fscore.append(model_result['F1 Score']['SVC'])
      sgd_fscore.append(model_result['F1 Score']['SGDClassifier'])
      nuSVC_fscore.append(model_result['F1 Score']['NuSVC'])
      logistic_reg_fscore.append(model_result['F1 Score']['LogisticRegression'])
      extraTrees_fscore.append(model_result['F1 Score']['ExtraTreesClassifier'])
      gaussianNB_fscore.append(model_result['F1 Score']['GaussianNB'])
      kNeighbours_fscore.append(model_result['F1 Score']['KNeighborsClassifier'])
      labelPropagation_fscore.append(model_result['F1 Score']['LabelPropagation'])
      lgbm_fscore.append(model_result['F1 Score']['LGBMClassifier'])
      bagging_fscore.append(model_result['F1 Score']['BaggingClassifier'])
  print("Done with random_state : ", random_state)
```

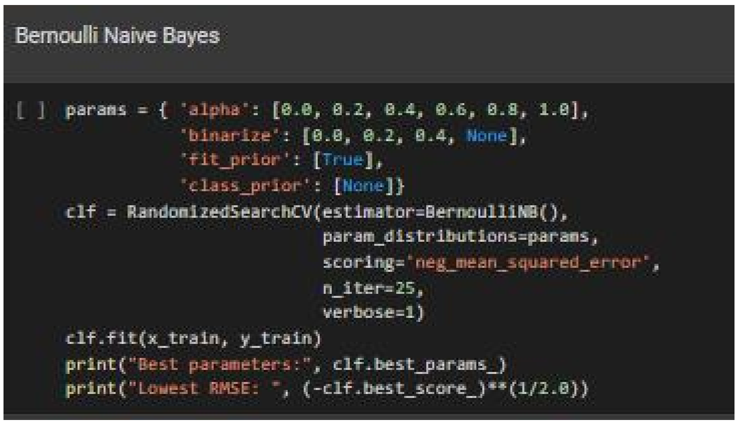
Code 12. LazyClassifier model training.

In Code 12, this code trains the lazyclassifier models through 5 folds of cross validation on 10 different random states. The metrics results from each loop cycle are appended into the list that will be used for calculating the metrics to determine the best performing models.


Multilayer Perceptron

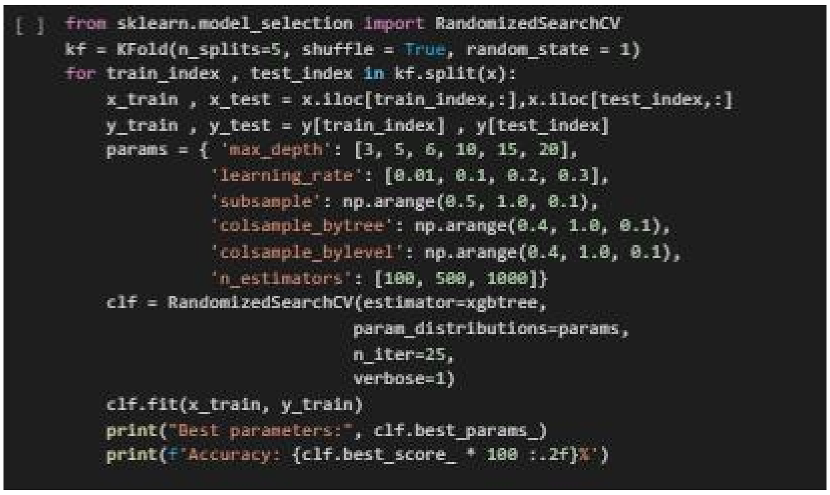
```
[ ] params = dict()
    params['hidden_layer_sizes'] = [150, 50, 100]
    params['activation'] = ['identity', 'logistic', 'tanh', 'relu']
    params['solver'] = ['lbfgs', 'sgd', 'adam']
    params['alpha'] = [0.0001]
    #params['batch_size'] = [16, 32, 64, 128]
    #params['learning_rate'] = ('constant', 'invscaling', 'adaptive')        #used only for sgd
    #params['learning_rate_init'] = ('identity', 'logistic', 'tanh', 'relu')  #used only for sgd and adam
    #params['power_t'] = ('identity', 'logistic', 'tanh', 'relu')
    #params['shuffle'] = (True, False)
    #params['random_state'] = ('none')                                         #used only for sgd and adam
    #params['tol'] = ('identity', 'logistic', 'tanh', 'relu')                  #used when learning_rate is adaptive
    #params['verbose'] = ('identity', 'logistic', 'tanh', 'relu')              #used to print progress
    #params['warm_start'] = (True, False)
    #params['max_iter'] = (100, 200, 500, 1000)
    #params['momentum'] = ('identity', 'logistic', 'tanh', 'relu')             #used only for sgd
    #params['nesterovs_momentum'] = ('identity', 'logistic', 'tanh', 'relu')  #used only for sgd and momentum > 0
    #params['early_stopping'] = ('identity', 'logistic', 'tanh', 'relu')      #used only for adam
    #params['beta_1'] = ('identity', 'logistic', 'tanh', 'relu')               #used only for adam
    #params['beta_2'] = ('identity', 'logistic', 'tanh', 'relu')               #used only for adam
    #params['epsilon'] = ('identity', 'logistic', 'tanh', 'relu')              #used only for adam
    #params['n_iter_no_change'] = ('identity', 'logistic', 'tanh', 'relu')    #used only for sgd and adam
    #params['max_fun'] = ('identity', 'logistic', 'tanh', 'relu')              #used only for lbfgs

    # define the search
    search = BayesSearchCV(estimator=MLPClassifier(), search_spaces=params, cv=kf)

    # perform the search
    search.fit(x_train, y_train)

    # report the best result
    print(search.best_score_)
    print(search.best_params_)
```

Code 13. Optimizing Multilayer Perceptron.


Bernoulli Naive Bayes

```
[ ] params = { 'alpha': [0.0, 0.2, 0.4, 0.6, 0.8, 1.0],
               'binarize': [0.0, 0.2, 0.4, None],
              'fit_prior': [True],
              'class_prior': [None]}
    clf = RandomizedSearchCV(estimator=BernoulliNB(),
                             param_distributions=params,
                             scoring='neg_mean_squared_error',
                             n_iter=25,
                             verbose=1)
    clf.fit(x_train, y_train)
    print("Best parameters:", clf.best_params_)
    print("Lowest RMSE: ", (-clf.best_score_)**(1/2.0))
```

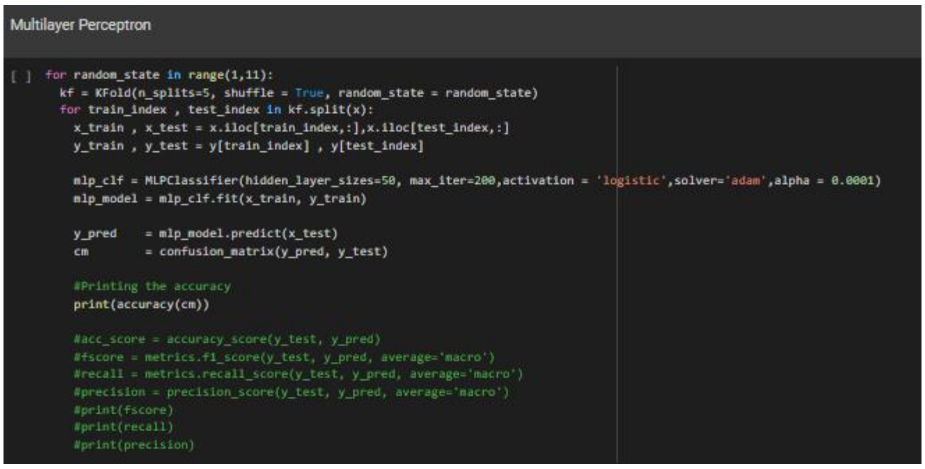
Code 14. Optimizing Bernoulli Naive Bayes.



```
[ ] from sklearn.model_selection import RandomizedSearchCV
    kf = KFold(n_splits=5, shuffle = True, random_state = 1)
    for train_index , test_index in kf.split(x):
        x_train , x_test = x.iloc[train_index,:],x.iloc[test_index,:]
        y_train , y_test = y[train_index] , y[test_index]
        params = { 'max_depth': [3, 5, 6, 10, 15, 20],
                   'learning_rate': [0.01, 0.1, 0.2, 0.3],
                   'subsample': np.arange(0.5, 1.0, 0.1),
                   'colsample_bytree': np.arange(0.4, 1.0, 0.1),
                   'colsample_bylevel': np.arange(0.4, 1.0, 0.1),
                   'n_estimators': [100, 500, 1000]}
        clf = RandomizedSearchCV(estimator=xgbtree,
                                 param_distributions=params,
                                 n_iter=25,
                                 verbose=1)
        clf.fit(x_train, y_train)
        print("Best parameters:", clf.best_params_)
        print(f'Accuracy: {clf.best_score_ * 100 :.2f}%')
```

Code 15. Optimizing XGBoost.

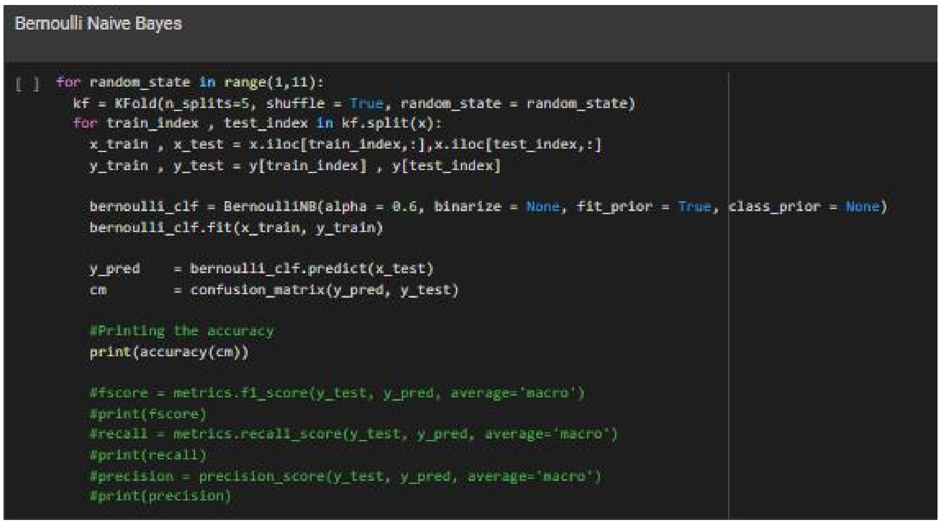
In Code 13, Code 14 and Code 15, the codes are for implementing the hyperparameter tuning.



Multilayer Perceptron

```
[ ] for random_state in range(1,11):
      kf = KFold(n_splits=5, shuffle = True, random_state = random_state)
      for train_index , test_index in kf.split(x):
        x_train , x_test = x.iloc[train_index,:],x.iloc[test_index,:]
        y_train , y_test = y[train_index] , y[test_index]

        mlp_clf = MLPClassifier(hidden_layer_sizes=50, max_iter=200,activation = 'logistic',solver='adam',alpha = 0.0001)
        mlp_model = mlp_clf.fit(x_train, y_train)

        y_pred    = mlp_model.predict(x_test)
        cm        = confusion_matrix(y_pred, y_test)

        #Printing the accuracy
        print(accuracy(cm))

        #acc_score = accuracy_score(y_test, y_pred)
        #fscore = metrics.f1_score(y_test, y_pred, average='macro')
        #recall = metrics.recall_score(y_test, y_pred, average='macro')
        #precision = precision_score(y_test, y_pred, average='macro')
        #print(fscore)
        #print(recall)
        #print(precision)
```

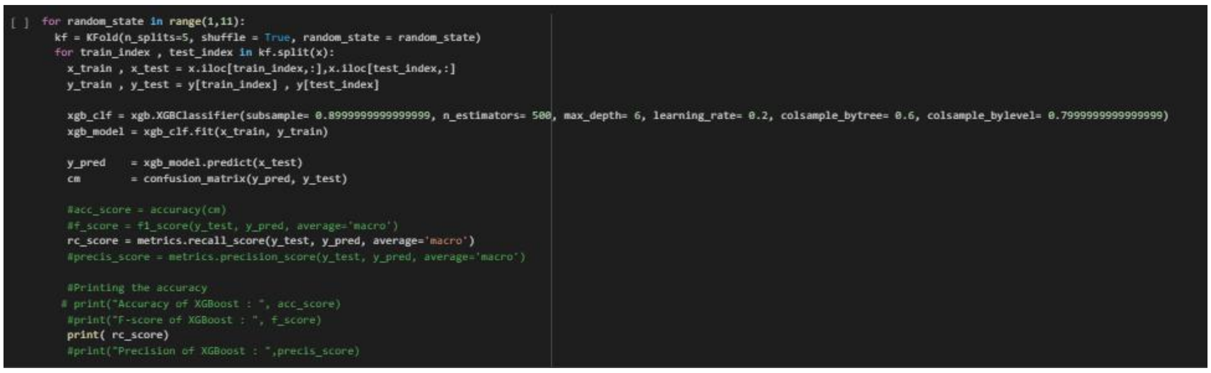
Code 16. Evaluate optimized MLP.



Bernoulli Naive Bayes

```
[ ] for random_state in range(1,11):
      kf = KFold(n_splits=5, shuffle = True, random_state = random_state)
      for train_index , test_index in kf.split(x):
        x_train , x_test = x.iloc[train_index,:],x.iloc[test_index,:]
        y_train , y_test = y[train_index] , y[test_index]

        bernoulli_clf = BernoulliNB(alpha = 0.6, binarize = None, fit_prior = True, class_prior = None)
        bernoulli_clf.fit(x_train, y_train)

        y_pred    = bernoulli_clf.predict(x_test)
        cm        = confusion_matrix(y_pred, y_test)

        #Printing the accuracy
        print(accuracy(cm))

        #fscore = metrics.f1_score(y_test, y_pred, average='macro')
        #print(fscore)
        #recall = metrics.recall_score(y_test, y_pred, average='macro')
        #print(recall)
        #precision = precision_score(y_test, y_pred, average='macro')
        #print(precision)
```

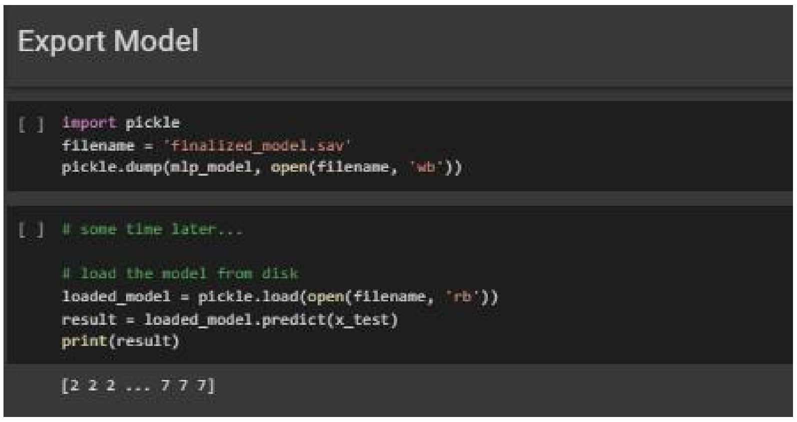
Code 17. Evaluate optimized Bernoulli Naive Bayes.



```
[ ] for random_state in range(1,11):
      kf = KFold(n_splits=5, shuffle = True, random_state = random_state)
      for train_index , test_index in kf.split(x):
        x_train , x_test = x.iloc[train_index,:],x.iloc[test_index,:]
        y_train , y_test = y[train_index] , y[test_index]

        xgb_clf = xgb.XGBClassifier(subsample= 0.8999999999999999, n_estimators= 500, max_depth= 6, learning_rate= 0.2, colsample_bytree= 0.6, colsample_bylevel= 0.7999999999999999)
        xgb_model = xgb_clf.fit(x_train, y_train)

        y_pred    = xgb_model.predict(x_test)
        cm        = confusion_matrix(y_pred, y_test)

        #acc_score = accuracy(cm)
        #f_score = f1_score(y_test, y_pred, average='macro')
        rc_score = metrics.recall_score(y_test, y_pred, average='macro')
        #precis_score = metrics.precision_score(y_test, y_pred, average='macro')

        #Printing the accuracy
       # print("Accuracy of XGBoost : ", acc_score)
        #print("F-score of XGBoost : ", f_score)
        print( rc_score)
        #print("Precision of XGBoost : ",precis_score)
```

Code 18. Evaluate optimized XGBoost.

Code 16, Code 17 and Code 18 shows the implementation of models with tuned parameters to provide results of optimized models.



Export Model

```
[ ] import pickle
    filename = 'finalized_model.sav'
    pickle.dump(mlp_model, open(filename, 'wb'))
```

```
[ ] # some time later...

    # load the model from disk
    loaded_model = pickle.load(open(filename, 'rb'))
    result = loaded_model.predict(x_test)
    print(result)

    [2 2 2 ... 7 7 7]
```

Code 19. Export model.

Code 19 shows the code for exporting the model. Pickle was used to export and load the models.