République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Benyoucef BENKHEDDA -Alger1-

Faculté des Sciences
Département Informatique

Thème

Projet d'apprentissage automatique (Machine Learning) basé sur le traitement d'un problème de Classification d'un jeu de données concernant L'Immunothérapie (Cancer de poumon)

Réalisé par

AOUNE Ramzi

2022/2023

Table of Content

I. Introduction :

Machine learning has emerged as a transformative field in healthcare, offering new possibilities for disease treatment and management. In particular, immunotherapy has gained significant attention as a promising approach in the fight against cancer. This report focuses on the application of machine learning techniques in the context of immunotherapy for lung cancer.

Lung cancer poses a significant threat to public health, accounting for a substantial number of cancer-related deaths worldwide. While traditional treatment methods such as surgery, chemotherapy, and radiation therapy have been used, they come with limitations and often result in adverse side effects. Immunotherapy presents an innovative strategy that harnesses the power of the immune system to target cancer cells specifically.

By employing machine learning algorithms, we aim to analyze immunotherapy data to uncover patterns, predict treatment outcomes, and enhance patient care. This project explores various machine learning techniques including logistic regression, neural networks, support vector machines (SVM), and K-nearest neighbors (KNN) to gain insights into the effectiveness of immunotherapy.

Our objective is to identify key factors influencing treatment response, elucidate underlying mechanisms, and improve patient outcomes. Through the integration of machine learning into immunotherapy, personalized treatment strategies can be developed, leading to improved therapeutic success rates and enhanced patient well-being. By leveraging large-scale datasets and advanced algorithms, we can extract valuable information from complex immunotherapy data, thereby aiding in clinical decision-making.

This report provides a comprehensive overview of our project, detailing the steps undertaken, methodologies employed, and the findings obtained from the analysis of the immunotherapy dataset. The results obtained through machine learning analysis have the potential to contribute to ongoing advancements in immunotherapy and shape the future of lung cancer treatment. However, it is important to acknowledge the challenges and limitations associated with machine learning, including robust data collection, model validation, and interpretability.

The collaboration between healthcare professionals and machine learning experts offers a promising path towards more effective and personalized immunotherapy approaches, instilling hope in lung cancer patients.

II. Theme Presentaion :

1. What is Immunotherapy ? :

Immunotherapy is a groundbreaking approach in the field of medical treatment that harnesses the power of the immune system to fight against diseases, particularly cancer. It revolves around the principle of utilizing the body's natural defense mechanisms to specifically target and eliminate abnormal cells, including cancer cells. Unlike traditional treatments such as chemotherapy or radiation therapy, which directly attack cancer cells but can also harm healthy cells, immunotherapy focuses on bolstering the immune system's ability to recognize and destroy cancerous cells. It achieves this by stimulating or enhancing the body's immune response through various methods.

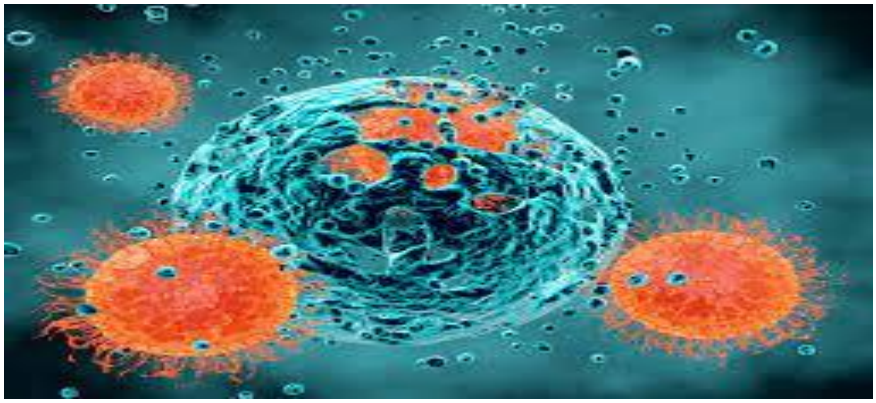

Figure 1 : Scientific photography for explaining immunotherapy

Immunotherapy encompasses a range of strategies, including the use of immune checkpoint inhibitors, monoclonal antibodies, cancer vaccines, adoptive cell transfer, and cytokine therapy. Immune checkpoint inhibitors, for instance, work by inhibiting certain proteins that limit the immune system's response, thereby unleashing its full potential to recognize and attack cancer cells. Monoclonal antibodies, on the other hand, are designed to bind to specific targets on cancer cells, flagging them for destruction by the immune system. Cancer vaccines aim to stimulate the immune system's recognition and response to cancer cells, while adoptive cell transfer involves genetically modifying immune cells to better target cancerous cells. Cytokine therapy utilizes specific proteins or molecules that regulate and enhance immune responses.

Immunotherapy has revolutionized cancer treatment and holds immense promise for patients. It offers the potential for long-term remission, fewer side effects compared to traditional treatments, and improved overall survival rates. Furthermore, it has shown effectiveness in various cancer types, including melanoma, lung cancer, bladder cancer, and more. The success of immunotherapy lies in its ability to utilize the body's intrinsic defense mechanisms, providing a more targeted and personalized treatment approach.

However, it is important to note that while immunotherapy has demonstrated remarkable results for some patients, it may not be effective for everyone or for every type of cancer.

Research is ongoing to further enhance its efficacy, expand its applications, and identify biomarkers that can predict response to immunotherapy. Additionally, managing potential immune-related adverse effects and optimizing treatment protocols are areas of active investigation.

Overall, immunotherapy represents a paradigm shift in cancer treatment, offering new hope for patients and healthcare providers. It has the potential to transform the way we approach and combat various diseases by leveraging the remarkable power of the immune system. Continued research and development in this field hold the promise of further advancements and increased success in the fight against cancer and other diseases.

2. Menaces that immunotherapy has able to do :

Immunotherapy, as a treatment approach for various diseases, including cancer and autoimmune disorders, has shown promise in improving patient outcomes. While immunotherapy has demonstrated significant benefits, it is essential to consider potential risks and challenges associated with this treatment. Here are some of the potential threats or challenges that immunotherapy may pose:

- Immune-related adverse events (irAEs): Immunotherapy works by stimulating the immune system, which can result in immune-related adverse events. These can include inflammation of organs or tissues, such as the lungs, liver, colon, or endocrine glands. These adverse events can range from mild to severe and require close monitoring and management.

- Autoimmune reactions: Immunotherapy treatments can potentially trigger autoimmune reactions, where the immune system attacks healthy cells and tissues. This can lead to autoimmune disorders such as thyroiditis, colitis, or pneumonitis.

- Lack of response: While immunotherapy has shown remarkable success in some patients, it does not work for everyone. Some individuals may not respond to immunotherapy, resulting in limited efficacy of the treatment for certain individuals or cancer types.

- Development of resistance: Similar to other cancer treatments, immunotherapy can face the challenge of the development of resistance over time. Cancer cells may evolve and find ways to evade the immune system's response, leading to treatment resistance and disease progression.

- High cost: Immunotherapy treatments can be expensive, making them less accessible to some individuals. The cost of these therapies, including monoclonal antibodies or adoptive cell therapies, can pose a financial burden on patients and healthcare systems.

- Limited applicability: While immunotherapy has demonstrated effectiveness in certain types of cancer, its applicability may be limited to specific subtypes or stages of the disease. Extending the use of immunotherapy to a broader range of cancer types and patient populations remains an area of ongoing research and development.

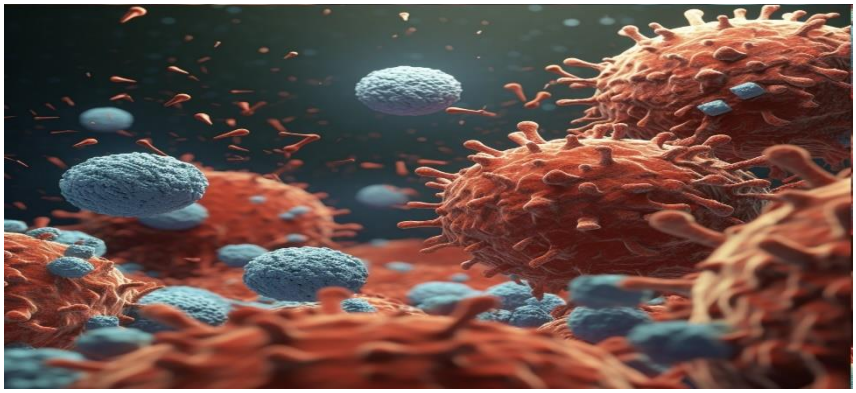

Figure 2 : Scientific photography for explaining its menaces

3. Consequences :

Immunotherapy has brought about significant consequences and transformative outcomes in the field of cancer treatment. These consequences have had a profound impact on patients, healthcare systems, and the overall landscape of cancer care.

One of the primary consequences of immunotherapy is the potential for improved treatment outcomes. By harnessing the power of the immune system, immunotherapy has demonstrated remarkable efficacy in certain types of cancer. It has led to increased survival rates, extended periods of remission, and enhanced quality of life for patients. This breakthrough treatment approach offers new hope and options for individuals facing cancer diagnoses.

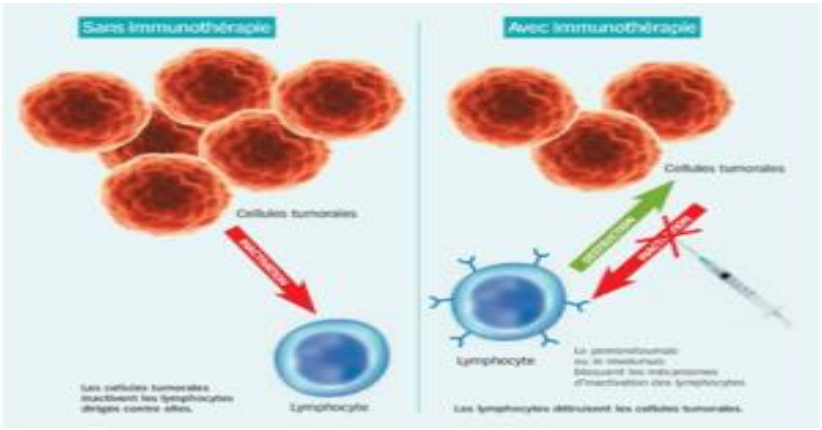

Figure 3 : Scientific schema explains virus infection & transmission

Additionally, immunotherapy has the potential to revolutionize cancer care by providing personalized treatment strategies. It takes into account the unique genetic makeup of each patient and the specific characteristics of their tumor. This personalized approach allows for targeted therapies tailored to individual needs, resulting in more effective treatment outcomes and reduced side effects compared to traditional therapies.

Immunotherapy has also contributed to advancements in our understanding of cancer biology and the immune system. The research and development associated with immunotherapeutic interventions have shed light on the complex interactions between cancer cells and the immune system. This knowledge has paved the way for further discoveries and the development of novel therapies.

Another consequence of immunotherapy is the potential for fewer treatment-related side effects. Unlike conventional treatments such as chemotherapy, which can cause significant toxicity and harm to healthy cells, immunotherapy specifically targets cancer cells while sparing healthy tissues. This targeted approach minimizes the occurrence of adverse effects, leading to a better quality of life for patients during and after treatment.

4. How to overcome it ? :

Overcoming immunotherapy challenges and maximizing its benefits require a comprehensive approach that encompasses various aspects of cancer care. Here are 3 strategies and interventions that can contribute to effectively overcoming the obstacles associated with immunotherapy:

Enhancing early detection and diagnosis of cancer: Timely detection plays a crucial role in improving treatment outcomes. Investing in screening programs and promoting awareness can lead to early diagnosis, enabling the timely initiation of immunotherapeutic interventions.

Expanding access to immunotherapy: Ensuring equitable access to immunotherapy is essential. This involves addressing barriers such as high treatment costs, limited availability in certain regions, and disparities in healthcare access. Collaboration between healthcare systems, policymakers, and pharmaceutical companies is needed to expand access.

Conducting extensive research: Continued research efforts are necessary to expand our understanding of immunotherapy mechanisms, identify predictive biomarkers, and develop more effective treatment strategies. Investing in scientific research and clinical trials can drive innovation and improve patient outcomes.

.

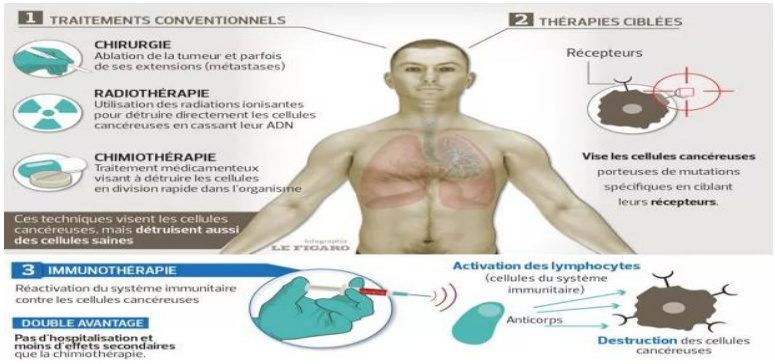

Figure 4 : Procedures have taken to overcoming this cancer

III. Classification task :

1. Representation of the dataset :

I choosed the dataset called « immunotherapy » it contains information about wart treatement result for 90 patients using immunotherapy, which The dataset is in the form of an Excel file named "Immunotherapy.xlsx".
Link to dataset : https://archive.ics.uci.edu/dataset/428/immunotherapy+dataset

1.1. Dataset information :

The dataset contains information about wart treatment results of 90 patients using immunotherapy

1.2. Relevant information about the dataset :

Subject Area: Life
Associated Tasks: Classification
Attribute Type: Integer, Real
Number of Instances: 90
Number of Attributes: 8

1.3. Information about each attribute in the dataset :

The dataset consists of the following attributes:

a) **Sex:** Represents the gender of the patient.
   Data Type: Integer
   Values: 1 (Male), 2 (Female)
b) **Age:** Represents the age of the patient.
   Data Type: Integer
   Values: Varying integer values
c) **Time:** Represents the duration of the treatment in months.
   Data Type: Real (Decimal)
   Values: Varying real values
d) **Number_of_Warts:** Represents the number of warts the patient had.
   Data Type: Integer
   Values: Varying integer values
e) **Type:** Represents the type of wart.
   Data Type: Integer
   Values: Varying integer values
f) **Area:** Represents the area of the wart.
   Data Type: Real (Decimal)
   Values: Varying real values

g) **Induration_diameter:** Represents the diameter of induration (hardening/swelling) caused by the wart.

Data Type: Integer

Values: Varying integer values

h) **Result_of_Treatment:** Represents the result of the wart treatment.

Data Type: Integer

Values: 0 (Failure), 1 (Success)

1.4. What we are using ? :

Inputs : i'm using Sex, Age, Time, Number_of_Warts, Type, Area, Induration diameter as inputs because that what we need to detect the failing or successing of Result_of_Treatement

Output/Target : i'm using Result_of_Treatement as output because it can tell as if this result is positive (Success) or negative (Failure)

1.5. Why this dataset ? :

This dataset contains 90 instances which was selected because it aligns with the research objective of the project. Immunotherapy is a rapidly advancing field in healthcare, and analyzing a dataset related to immunotherapy allows us to gain insights into the effectiveness and outcomes of this treatment approach.

2. Implementations to the dataset :

By employing machine learning algorithms, we aim to analyze immunotherapy data to uncover patterns, predict treatment outcomes, and enhance patient care.

So for that, this project explores various machine learning supervised techniques including logistic regression, neural networks, support vector machines (SVM), K-nearest neighbors (KNN), and Regularisation(Normalisation) & Cross Validation that i implement them to this dataset to gain insights into the effectiveness of immunotherapy.

2.1. Logistic Regression :

2.1.1. Importing libraries :

I import all libraries which used to implement out dataset by Logistic Regression technique, as the following picture below :

```python
#The imports of various libraries
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
```

Figure 5 : Code of importing libraries used to implement by Logistic Regression

2.1.2. Loading data :

To continue and reach the results of following steps executions, we must read our dataset and load it by pd.read_excel() and not by pd.read_csv cause we have an .xlsx as a type of folder's extension

```python
#Load the data
data = pd.read_excel('Immunotherapy.xlsx')
```

Figure 6 : Code of loading the data

2.1.3. Preprocessing phase :

One of the most important phases which helps us to clean useless caracters, dupplicated rows, multiple spaces which all of thoses represent as a part of aberrant and missing values that have the ability to damage our dataset

```python
#Remove leading and trailing white spaces from string columns
def deleteUselessCaracters(self):
    self.data = self.data.apply(lambda x: x.str.strip() if x.dtype == "object" else x)

#Replace multiple spaces with a single space in string columns
def remove_multiple_spaces(self):
    self.data = self.data.apply(lambda x: x.str.replace('\s+', ' ') if x.dtype == "object" else x)

#Replace double spaces with a single space in string columns
def deleteDoubleSpace(self):
    self.data = self.data.apply(lambda x: x.str.replace('  ', ' ') if x.dtype == "object" else x)
```

Figure 7 : A part of preprocessing data code

2.1.4. Training phase :

Important phase which i showed the size of my dataset before and after preprocess it to see any changes on it

```python
#Training the data
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

#Display the size of data before removing duplicated rows
print("Shape of data before deleting duplicated rows:", data.shape)

#Deleting duplicated rows
preprocess.delete()

#Display the size of data after removing duplicated rows
print("Shape of data after deleting duplicated rows:", data.shape)
```

```
Shape of data before deleting duplicated rows: (90, 8)
Shape of data after deleting duplicated rows: (90, 8)
```

Figure 8 : Code of data training phase

2.1.5. Testing phase :

In this phase, i've made a test to my model by calculate its mean accuracy and showing its matrix confusion with the code as following below :

```
#Testing the data by calculating mean accuracy
y_pred = logreg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Mean Accuracy:", accuracy)

#Displaying and ploting confusion matrix
confusion = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
sns.heatmap(confusion, annot=True, cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

Figure 9 : code of data testing phase

Results : this is the mean accuracy and matrix confusion of my model :

```
Mean Accuracy: 0.8333333333333334
Confusion Matrix:
```

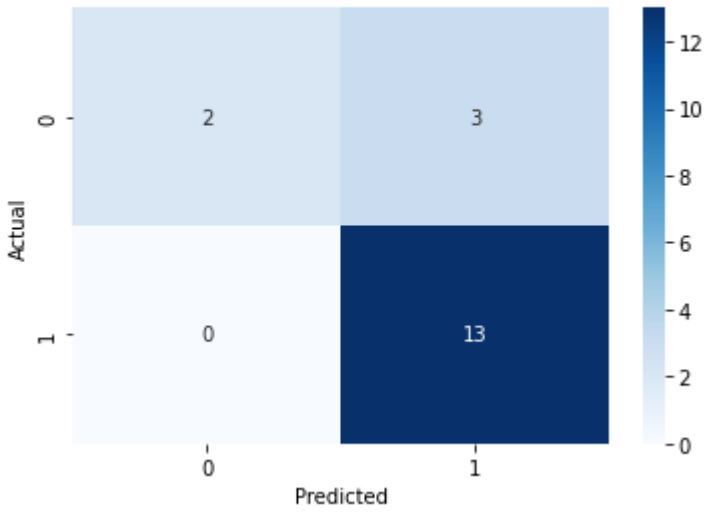

Figure 10 : Results of data testing phase

Which the values of the matrix representing as following :

2 : Represents the values of **True positive**

3 : Represents the values of **False negative**

0 : Represents the values of **False positive**

13 : Represents the values of **True negative**

2.1.6. Ploting :

Here, i ploted my data by showing markersized points of my matrix confusion that has been shown before :

```
#Ploting the data
plt.spy(confusion, markersize=5)
plt.show()
```

Figure 11 : Code representing the ploting data

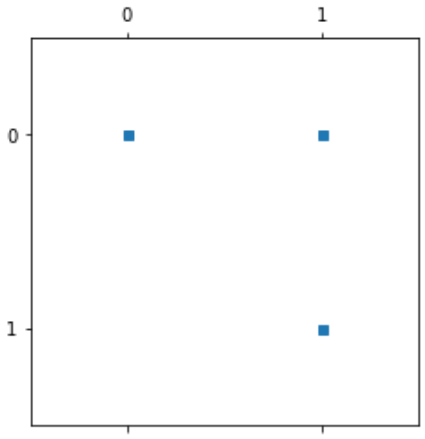

Figure 12 : Results of ploting data

2.1.7. Making prediction :

I've made a prediction for the result of treatement which it can be successful result (positive) or failure (negative)

```
# Make the prediction for 'Result_of_Treatment_positive' (1)
positive_result_of_treatment = logreg.predict([X_test.iloc[0]])  # Input the relevant features for prediction (e.g.,

# Make the prediction for 'Result_of_Treatment_negative' (0)
negative_result_of_treatment = logreg.predict([X_test.iloc[-1]])  # Input a different relevant sample for prediction

#Result of predictions
print("Prediction for 'Result_of_Treatment_positive' (1):", positive_result_of_treatment)
print("Prediction for 'Result_of_Treatment_negative' (0):", negative_result_of_treatment)
```

Figure 13 : Code of making a predicion

```
Prediction for 'Result_of_Treatment_positive' (1): [1]
Prediction for 'Result_of_Treatment_negative' (0): [0]
```

Figure 14 : Results of making a prediction

2.1.8. Saving the model :

I saved my model using Joblib ,It provides utilities for saving and loading Python objects , so we used it to save my model

```
joblib.dump(logreg, 'myModel.joblib')
```

```
['myModel.joblib']
```

Figure 14 : Saving the model

2.2. Neural Network :

2.2.1. Importing libraries :

I import all libraries which used to implement out dataset by Neural Network technique, as the following picture below :

```
#The import of various libraries
import numpy as np
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
```

Figure 15 : Code of importing libraries used to implement by Neural Network

2.2.2. Loading data :

I add to loading data a separation into X which loads the features data and y which loads the target variable data

```
#Load the Data
data = pd.read_excel('Immunotherapy.xlsx')
X = data.iloc[:, :-1]  # Features
y = data.iloc[:, -1]   # Target variable
```

Figure 16 : Code of loading the data

2.2.3. Preprocessing phase :

In this phase, i attempt to clean my dataset by removing stop words :

```
def remove_stop(x):
    #Code to remove stop words from x
    stop_words = ['the', 'a', 'an', 'and', 'or', 'but', 'not']  # Example list of stop words
    x = ' '.join([word for word in x.split() if word.lower() not in stop_words])
    return x
```

Figure 17 : Function of removing stop words as a part of processing data

Then, i removed multiple spaces as a second step :

```
def remove_multiple_spaces(x):
    #Code to remove multiple spaces from x
    x = ' '.join(x.split())
    return x
```

Figure 18 : Function of removing multiple spaces as a part of processing data

Third and last step is the final preprocessing by applying the preverious functions of remove_stop and remove_multiple_spaces :

```
def preprocessing(data):
    #Preprocessing steps for data
    data['text'] = data['text'].apply(remove_stop)
    data['text'] = data['text'].apply(remove_multiple_spaces)
    return data
```

Figure 19 : Function of final preprocessing as a part of processing data

2.2.4. Hidden layers incrementation :

As we're knowing Neural Network is a huge importnat classification task in Machine Learning, so it can't be created without inclunding Hidden Layers as the main link which reach us to the target (output layer) starting from integer layers .

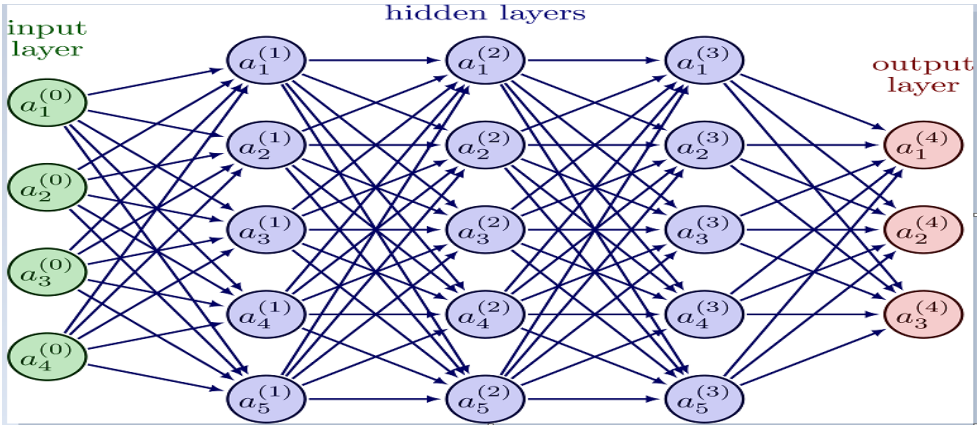

Figure 20 : An example of full Neural Network with usling three Hidden Layers

So in this phase, the objective is i'll create a function to select the optimal Hidden Layer, so i'll use three functions :

- One Hidden Layer : that have one hidden layer :

```
def one_hidden_layer(X_main, y_main, iter, noeuds_hidden_layers, alphas):
    #Create a MLPClassifier with one hidden layer
    model = MLPClassifier(hidden_layer_sizes=(noeuds_hidden_layers,), max_iter=iter, alpha=alphas)

    #Split the data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X_main, y_main, test_size=0.2, random_state=42)

    #Fit the model on the training data
    model.fit(X_train, y_train)

    #Predict on the test data
    y_pred = model.predict(X_test)

    #Calculate accuracy score
    accuracy = accuracy_score(y_test, y_pred)

    return accuracy
```

Figure 21 : A function which let me using my conecption of Neural Network with one Hidden Layer

- Two Hidden Layers : that have two hidden layers :

```
def two_hidden_layer(X_main, y_main, iter, noeuds_hidden_layers, alphas):
    #Create a MLPClassifier with two hidden layers
    model = MLPClassifier(hidden_layer_sizes=(noeuds_hidden_layers, noeuds_hidden_layers), max_iter=iter, alpha=alphas)

    #Split the data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X_main, y_main, test_size=0.2, random_state=42)

    #Fit the model on the training data
    model.fit(X_train, y_train)

    #Predict on the test data
    y_pred = model.predict(X_test)

    #Calculate accuracy score
    accuracy = accuracy_score(y_test, y_pred)

    return accuracy
```

Figure 22 : A function which let me using my conecption of Neural Network with two Hidden Layers

- Three Hidden Layers : that have three hidden layers :

```
def three_hidden_layer(X_main, y_main, iter, noeuds_hidden_layers, alphas):
    # Create a MLPClassifier with three hidden layers
    model = MLPClassifier(hidden_layer_sizes=(noeuds_hidden_layers, noeuds_hidden_layers, noeuds_hidden_layers)
                          max_iter=iter, alpha=alphas)

    #Split the data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X_main, y_main, test_size=0.2, random_state=42)

    #Fit the model on the training data
    model.fit(X_train, y_train)

    #Predict on the test data
    y_pred = model.predict(X_test)

    #Calculate accuracy score
    accuracy = accuracy_score(y_test, y_pred)

    return accuracy
```

Figure 23 : A function which let me using my conecption of Neural Network with three Hidden Layers

Choosing Hidden Layer : In this step i choose the Hidden Layer that i found it more optimal to use than the others according to the best accuracy :

```
def which_hidden_layer_we_choose(X_main, y_main, best_one_hidden_layer, best_two_hidden_layer, best_three_hidden_layer):
    best_accuracy = max(best_one_hidden_layer, best_two_hidden_layer, best_three_hidden_layer)
    if best_accuracy == best_one_hidden_layer:
        return "One Hidden Layer"
    elif best_accuracy == best_two_hidden_layer:
        return "Two Hidden Layers"
    elif best_accuracy == best_three_hidden_layer:
        return "Three Hidden Layers"
```

Figure 24 : A function which let me choose which Hidden Layer is more optimal according to accuracy

2.2.5. Main function :

In this final phase, i used the main function as a summary code of all preverious function that we did with incrementation values of iterations, nodes, alphas of each Hidden Layer used :

```
def main():
    #Load the dataset
    data = pd.read_csv('your_dataset.csv')

    #Preprocess the data
    preprocessed_data = preprocessing(data)

    #Extract features and labels
    X = preprocessed_data['text']
    y = preprocessed_data['label']

    #Hyperparameter tuning for one hidden layer
    best_one_hidden_layer = one_hidden_layer(X, y, iter=1000, noeuds_hidden_layers=10, alphas=0.0001)

    #Hyperparameter tuning for two hidden layers
    best_two_hidden_layer = two_hidden_layer(X, y, iter=1000, noeuds_hidden_layers=10, alphas=0.0001)

    #Hyperparameter tuning for three hidden layers
    best_three_hidden_layer = three_hidden_layer(X, y, iter=1000, noeuds_hidden_layers=10, alphas=0.0001)

    #Choose the best hidden layer configuration
    best_hidden_layer = which_hidden_layer_we_choose(X, y, best_one_hidden_layer, best_two_hidden_layer,
                                                     best_three_hidden_layer)

    print("Best Hidden Layer Configuration:", best_hidden_layer)
```

Figure 25 : The main function of Neural Network

2.3. Support Vector Machine (SVM) :

2.3.1. Importing libraries :

I import all libraries which used to implement out dataset by SVM technique, as the following picture below :

```
#The import of various libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

Figure 26 : Code of importing libraries used to implement by SVM

2.3.2. Dataset use :

First, while i started to use my dataset, i made the first step which is loading my data :

```
data = pd.read_excel('Immunotherapy.xlsx')
```

Figure 27 : Code of loading my data

Then, i displayed my first rows by this command as a second step :

```
data.head()
```

Figure 28 : Command of displaying my first rows of the data

And, there is the result :

	sex	age	Time	Number_of_Warts	Type	Area	induration_diameter	Result_of_Treatment
0	1	22	2.25	14	3	51	50	1
1	1	15	3.00	2	3	900	70	1
2	1	16	10.50	2	1	100	25	1
3	1	27	4.50	9	3	80	30	1
4	1	20	8.00	6	1	45	8	1

Figure 29 : Result of the command which shows a table displaying first 5 rows of the data

After, i display all of my dataset as a third step :

```
print(data)
```

Figure 30 : Command of displaying my first rows of the data

Then, the result is a full dataset with all instance of each integer variables & output variable, which each column represents the variables (features) and each row represents the instances.

The pictures as the following belows shows the result :

```
    sex   age    Time   Number_of_Warts   Type   Area   induration_diameter
0     1    22    2.25                14      3     51                    50
1     1    15    3.00                 2      3    900                    70
2     1    16   10.50                 2      1    100                    25
3     1    27    4.50                 9      3     80                    30
4     1    20    8.00                 6      1     45                     8
..  ...   ...     ...               ...    ...    ...                   ...
85    1    40    5.50                 8      3     69                     5
86    1    38    7.50                 8      2     56                    45
87    1    46   11.50                 4      1     91                    25
88    1    32   12.00                 9      1     43                    50
89    2    23    6.75                 6      1     19                     2

    Result_of_Treatment
0                     1
1                     1
2                     1
3                     1
4                     1
..                  ...
85                    1
86                    1
87                    0
88                    0
89                    1

[90 rows x 8 columns]
```

Figure 30 : Result of the command which shows the full dataset

Now, i display the size of this full dataset as fourth step :

```
print(f'numbers of rows and colums in this data set: {data.shape}')
```

Figure 31 : Command of displaying the size of the full dataset

The result is as it shows below :

```
numbers of rows and colums in this data set: (90, 8)
```

Figure 32 : Result of the size

In the fifth and final step, i show the information of my dataset :

```
#dataset informations
data.info()
```

Figure 33 : Command of displaying full informations

The result shows this :

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90 entries, 0 to 89
Data columns (total 8 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   sex                  90 non-null     int64
 1   age                  90 non-null     int64
 2   Time                 90 non-null     float64
 3   Number_of_Warts      90 non-null     int64
 4   Type                 90 non-null     int64
 5   Area                 90 non-null     int64
 6   induration_diameter  90 non-null     int64
 7   Result_of_Treatment  90 non-null     int64
dtypes: float64(1), int64(7)
memory usage: 5.8 KB
```

Figure 34 : Dataset information

2.3.3. Preprocessing phase :

```
#Preprocess the data
def preprocess(data):
    # Remove any rows with missing values
    data.dropna(inplace=True)

    #Separate the features (X) and target variable (y)
    X = data.iloc[:, :-1]
    y = data.iloc[:, -1]

    #Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    #Standardize the features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    return X_train, X_test, y_train, y_test

X_train, X_test, y_train, y_test = preprocess(data)
```

Figure 35 : Preprocessing phase code of SVM

2.3.4. Training phase :

I used to training the model by the command « model = svm.SVC(kernel='linear') » as the code shows with the result :

```
#Model Training
model = svm.SVC(kernel='linear')
model.fit(X_train, y_train)
```

```
SVC(kernel='linear')
```

Figure 36 : Code and result for model training of SVM

2.3.5. Testing phase :

```
#Model Testing
y_pred = model.predict(X_test)
```

Figure 37 : Code for model testing of SVM

2.3.6. Evaluation phase :

Here, i evaluate my SVM model by calculating its accuracy & matrix confusion :

Here is the code :

```
#Model Evaluation
accuracy = metrics.accuracy_score(y_test, y_pred)
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)

print("Accuracy:", accuracy)
print("Confusion Matrix:")
print(confusion_matrix)
```

Figure 38 : Code for model evaluation of SVM

And here is the result :

```
Accuracy: 0.7777777777777778
Confusion Matrix:
[[ 1  4]
 [ 0 13]]
```

Figure 39 : Result for model evaluation of SVM

2.3.7. Ploting :

This the most important phase that we have in SVM supervised technique learning, which i plot the training data & testing data for my SVM model :

There is the code which contains a set commandes used to plot what we need :

```
#We need to plot those training and testing data to make it easier for us to draw the Decision Boundary in the next step
def plot_data(X, y, title):
    plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], label="Class 0") #Class 0 points are represented in Blue
    plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], label="Class 1") #Class 1 points are represented in Orange
    plt.title(title)
    plt.legend()
    plt.show()

plot_data(X_train, y_train, "Training Data")
plot_data(X_test, y_test, "Testing Data")
```

Figure 40 : The main code of training & testing data plot

By the command : `plot_data(X_train, y_train, "Training Data")`

I ploted the Training Data which is this :

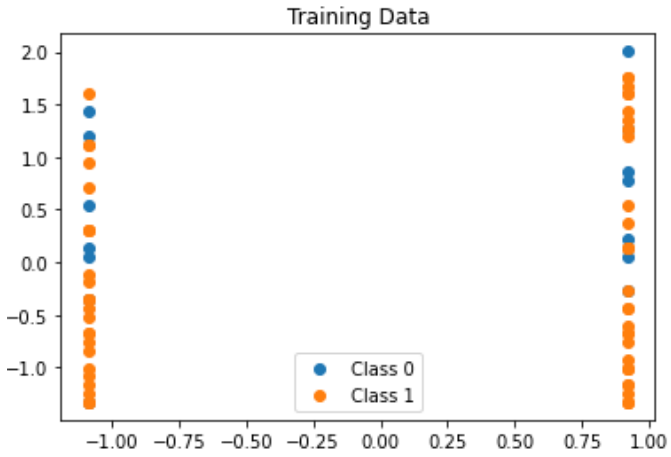

Figure 41 : Distributions of training data points according to x_train (x axis) and y_train (y axis)

By the command : `plot_data(X_test, y_test, "Testing Data")`

I ploted the Testing Data which is this :

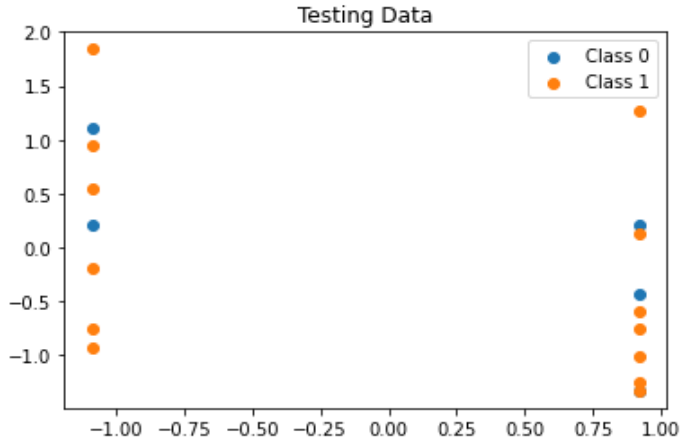

Figure 42 : Distributions of testing data points according to x_train (x axis) and y_train (y axis)

2.3.8. Decision boundary :

It's how we can trace a separate line between the two classes (class 0 and class 1) which they representing the distribution of data points in each of training and testing data, as the following picture below :

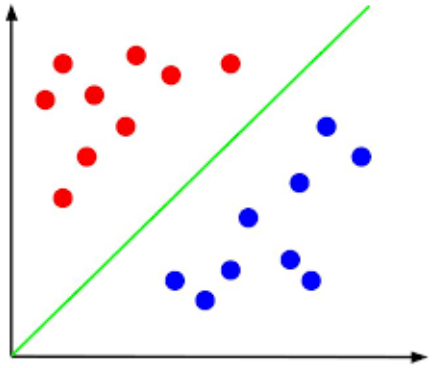

Figure 43 : simple example of SVM model which separates points by green line represents the decision boundary

And that's the code of decision boundary applied in our SVM model :

```
#Function of plotting the Decision Boundary that separates class 0 points from class 1 points in each of Training & Testing data
def plot_decision_boundary(model, X, y, title):
    # Set min and max values for the features
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    #Create a meshgrid of points
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                         np.arange(y_min, y_max, 0.1))

    #Make predictions on the meshgrid
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])

    #Reshape the predictions
    Z = Z.reshape(xx.shape)

    #Plot the decision boundary
    plt.contourf(xx, yy, Z, alpha=0.8)

    #Plot the data points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title(title)
    plt.show()
```

Figure 44 : The main code representing the function of plotting the Decision Boundary

2.4. K-Nearest Neighborhood :

2.4.1. Importing libraries :

I import all libraries which used to implement out dataset by KNN technique, as the following picture below :

```
#The import of various libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

Figure 45 : Code of importing libraries used to implement by KNN

2.4.2. Data representation :

First, while i started to represent my dataset, i made the first step which is loading my data :

```
#Loading the data
data=pd.read_excel('Immunotherapy.xlsx')
```

Figure 46 : Code of loading my data

Then, i described my data by this command as a second step :

```
#Describing the data
data.describe()
```

Figure 47 : Command of displaying my data

As consequence, the result shows this :

	sex	age	Time	Number_of_Warts	Type	Area	induration_diameter	Result_of_Treatment
count	90.000000	90.000000	90.000000	90.000000	90.000000	90.000000	90.000000	90.000000
mean	1.544444	31.044444	7.230556	6.144444	1.711111	95.700000	14.333333	0.788889
std	0.500811	12.235435	3.098166	4.212238	0.824409	136.614643	17.217707	0.410383
min	1.000000	15.000000	1.000000	1.000000	1.000000	6.000000	2.000000	0.000000
25%	1.000000	20.250000	5.000000	2.000000	1.000000	35.500000	5.000000	1.000000
50%	2.000000	28.500000	7.750000	6.000000	1.000000	53.000000	7.000000	1.000000
75%	2.000000	41.750000	9.937500	8.750000	2.000000	80.750000	9.000000	1.000000
max	2.000000	56.000000	12.000000	19.000000	3.000000	900.000000	70.000000	1.000000

Figure 48 : Data description

Third step, i calculated and i displayed the sum of my data :

```
#Calculate and display the sum of all attributes
data.isnull().sum()
```

Figure 49 : Command to display the sum of the data

The results of command's execution show this :

```
sex                 0
age                 0
Time                0
Number_of_Warts     0
Type                0
Area                0
induration_diameter 0
Result_of_Treatment 0
dtype: int64
```

Figure 50 : The results of the sum command's execution

Forth and final step, i calculated and i displayed the value counts which represent the value counts of each attribute (feature) of my dataset :

The command that i implemented for this, as the following below :

```
#Calculate and display value counts
data.value_counts()
```

Figure 51 : Command to display the value counts

The results of command's execution show this :

```
sex  age  Time   Number_of_Warts  Type  Area  induration_diameter  Result_of_Treatment
1    15   3.00   2                3     900   70                   1                     1
2    33   6.25   2                1     30    3                    1                     1
     28   11.00  3                3     91    6                    0                     1
          7.50   4                1     9     2                    1                     1
     26   10.50  6                1     50    9                    0                     1
                                                                                         ..
1    35   9.75   2                2     8     6                    1                     1
          8.75   10               2     69    7                    1                     1
          6.75   4                3     41    8                    1                     1
     34   5.00   7                3     64    7                    0                     1
2    56   11.75  7                1     31    50                   0                     1
Length: 90, dtype: int64
```

Figure 52 : The results of the value counts command's execution

2.4.3. Slicing phase :

To apply the KNN algorithm on the dataset, i must slice my data into two classes, the first class represents the data of input attributes, and the other class represents the data of output variable, which the code as following :

```
#Slicing the data into two classes
x=data.iloc[:,:-1].values #Class X which represent the data of input attributes
y=data.iloc[:,-1].values #Class y which represent the data of output attribute
```

Figure 53 : Code representing slicing data into two classes x and y

Now after slicing data, we display the data of class x by this command :

```
#Display Class X
print(x)
```

Figure 54 : Command show the data of class x

We will get this as results :

```
[[  1.    22.     2.25  14.     3.    51.    50.  ]
 [  1.    15.     3.     2.     3.   900.    70.  ]
 [  1.    16.    10.5    2.     1.   100.    25.  ]
 [  1.    27.     4.5    9.     3.    80.    30.  ]
 [  1.    20.     8.     6.     1.    45.     8.  ]
 [  1.    15.     5.     3.     3.    84.     7.  ]
 [  1.    35.     9.75   2.     2.     8.     6.  ]
 [  2.    28.     7.5    4.     1.     9.     2.  ]
 [  2.    19.     6.     2.     1.   225.     8.  ]
 [  2.    32.    12.     6.     3.    35.     5.  ]
 [  2.    33.     6.25   2.     1.    30.     3.  ]
 [  2.    17.     5.75  12.     3.    25.     7.  ]
 [  2.    15.     1.75   1.     2.    49.     7.  ]
 [  2.    15.     5.5   12.     1.    48.     7.  ]
 [  2.    16.    10.     7.     1.   143.     6.  ]
 [  2.    33.     9.25   2.     2.   150.     8.  ]
 [  2.    26.     7.75   6.     2.     6.     5.  ]
 [  2.    23.     7.5   10.     2.    43.     3.  ]
 [  2.    15.     6.5   19.     1.    56.     7.  ]
 [  2.    26.     6.75   2.     1.     6.     6.  ]
 [  1.    22.     1.25   3.     3.    47.     3.  ]
 [  2.    19.     2.25   2.     1.    60.     7.  ]
 [  2.    26.    10.5    6.     1.    50.     9.  ]
 [  1.    25.     5.75   2.     1.   300.     7.  ]
 [  2.    17.    11.25   4.     3.    70.     7.  ]
 [  1.    27.     5.     2.     1.    20.     5.  ]
 [  2.    24.     4.75  10.     3.    30.    45.  ]
 [  1.    15.    11.     6.     1.    30.    25.  ]
```

Figure 55 : Show the data of class x

By the same action, we also display the data of class y by this command :

```
#Display Class y
print(y)
```

Figure 56 : Command show the data of class y

We will get this as results :

```
[1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 1 1 1 1 1 0 0
 1 1 0 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 0 1 1 0 1
 1 1 1 1 1 0 1 1 0 1 1 1 1 0 0 1]
```

Figure 57 : Show the data of class y

2.4.4. KNN using K-Fold :

There are many algorithms that have been used with KNN supervised learning technique to split the dataset into a set or a group of classes which lead us to see the training data for each one of them.

Cause there are a lots of algorithms, i used to choose K-Fold algorithm which slice the dataset into K-mini classes with the same shape for each one of them.

To use the implementation of KNN by K-Fold algorithm, we have two steps that we must do :

First step, we stratify our algorithm by deviding our dataset in K number of classes with the same shape, we have the permission to choose the number of classes. As a choice, i chosed to split my data into 5 training datas with the same shape, as the following code below :

```
#Stratify the algorithm K-Fold which split the data into K mini classes
from sklearn.model_selection import StratifiedKFold
sf=StratifiedKFold(n_splits=5,shuffle=True,random_state=0)
sf.get_n_splits(x_data)
print(sf)
```

Figure 58 : Stratify the algorithm K-Fold

The result of the code shows this :

```
StratifiedKFold(n_splits=5, random_state=0, shuffle=True)
```

Figure 59 : The result of code's execution

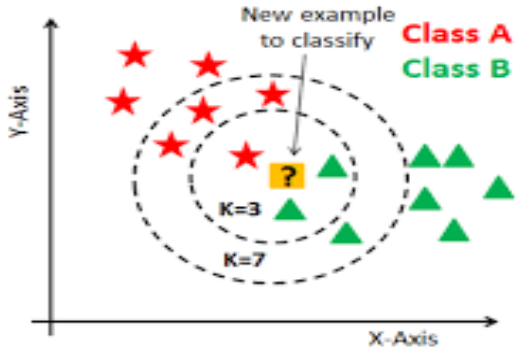

Figure 60 : An example of KNN using K-Fold

Second step, i implement the KNN supervised learning technique with using the stratified algorithm K-Fold that i've used to my dataset by using the code as the following below :

```
for train_index,test_index in sf.split(x_data,y_data):
 print("TRAIN:", train_index, "TEST:", test_index)
 x_train, x_test = x_data[train_index], x_data[test_index]
 y_train, y_test = y_data[train_index], y_data[test_index]


 # Define the algorihm
 from sklearn.neighbors import KNeighborsClassifier
 classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
 classifier.fit(x_train,y_train)
 from sklearn.model_selection import cross_val_score
 scores=cross_val_score(classifier,x_train,y_train,cv=sf)
 print(scores)
 print(np.mean(scores)*100)
```

Figure 61 : Implementation the KNN using K-Fold to the dataset

The result of the code's execution gave us this :

```
TRAIN: [  0   1   2   3   4   5   7   8   9  10  11  12  13  14  15  16  17  18
  19  20  23  24  26  27  28  31  32  34  35  36  37  38  39  40  41  42
  45  46  47  48  49  50  51  53  55  56  57  58  61  62  63  64  65  67
  68  70  72  73  75  76  78  79  80  81  82  84  85  86  87  88  89  90
  91  92  93  94  96  97  99 101 102 104 105 106 107 108 109 110 111 112
 113 114 115 116 117 119 120 122 123 124 125 126 127 128 129 130 131 133
 134 135 137 139 140] TEST: [  6  21  22  25  29  30  33  43  44  52  54  59  60  66  69  71  74  77
  83  95  98 100 103 118 121 132 136 138 141]
[0.52173913 0.73913043 0.47826087 0.59090909 0.72727273]
61.14624505928854
TRAIN: [  2   4   5   6   7   8   9  10  11  12  14  15  16  17  18  19  20  21
  22  23  24  25  26  27  28  29  30  31  32  33  34  36  37  38  39  40
  41  43  44  45  47  48  49  50  52  54  57  59  60  62  64  65  66  67
  68  69  70  71  72  73  74  75  77  79  80  83  84  85  86  88  89  90
  93  95  96  98 100 101 102 103 104 105 106 107 108 109 110 111 112 113
 114 115 116 117 118 119 121 122 124 127 128 129 130 131 132 133 134 135
 136 137 138 139 141] TEST: [  0   1   3  13  35  42  46  51  53  55  56  58  61  63  76  78  81  82
  87  91  92  94  97  99 120 123 125 126 140]
[0.65217391 0.65217391 0.60869565 0.68181818 0.68181818]
65.53359683794466
TRAIN: [  0   1   2   3   5   6   7   9  10  11  12  13  17  18  19  20  21  22
  23  24  25  27  28  29  30  31  32  33  35  36  37  38  39  40  42  43
  44  45  46  48  49  51  52  53  54  55  56  57  58  59  60  61  63  64
  65  66  67  69  71  73  74  75  76  77  78  81  82  83  84  85  86  87
  89  90  91  92  94  95  96  97  98  99 100 101 102 103 104 105 107 110
 111 113 115 116 118 119 120 121 122 123 125 126 127 129 131 132 133 134
 135 136 138 139 140 141] TEST: [  4   8  14  15  16  26  34  41  47  50  62  68  70  72  79  80  88  93
 106 108 109 112 114 117 124 128 130 137]
[0.82608696 0.69565217 0.60869565 0.65217391 0.68181818]
69.28853754940711
TRAIN: [  0   1   3   4   6   7   8  10  12  13  14  15  16  18  19  21  22  23
  25  26  28  29  30  33  34  35  40  41  42  43  44  45  46  47  49  50
  51  52  53  54  55  56  57  58  59  60  61  62  63  64  66  67  68  69
  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  87  88
  91  92  93  94  95  96  97  98  99 100 101 103 106 107 108 109 110 112
 113 114 115 116 117 118 120 121 123 124 125 126 127 128 129 130 131 132
 134 136 137 138 140 141] TEST: [  2   5   9  11  17  20  24  27  31  32  36  37  38  39  48  65  86  89
  90 102 104 105 111 119 122 133 135 139]
[0.69565217 0.69565217 0.56521739 0.65217391 0.59090909]
63.99209486166008
TRAIN: [  0   1   2   3   4   5   6   8   9  11  13  14  15  16  17  20  21  22
  24  25  26  27  29  30  31  32  33  34  35  36  37  38  39  41  42  43
  44  46  47  48  50  51  52  53  54  55  56  58  59  60  61  62  63  65
  66  68  69  70  71  72  74  76  77  78  79  80  81  82  83  86  87  88
  89  90  91  92  93  94  95  97  98  99 100 102 103 104 105 106 108 109
 111 112 114 117 118 119 120 121 122 123 124 125 126 128 130 132 133 135
 136 137 138 139 140 141] TEST: [  7  10  12  18  19  23  28  40  45  49  57  64  67  73  75  84  85  96
 101 107 110 113 115 116 127 129 131 134]
[0.73913043 0.73913043 0.73913043 0.69565217 0.77272727]
73.71541501976286
```

Figure 62 : Results of KNN with using K-Fold

2.5. Regularisation & Cross Validation :

Are one of the supervised learning techniques that used to normalise the dataset into small valors of variables which are similar to each other

Here is an example of Regularisaition or it's called as Normalisation :

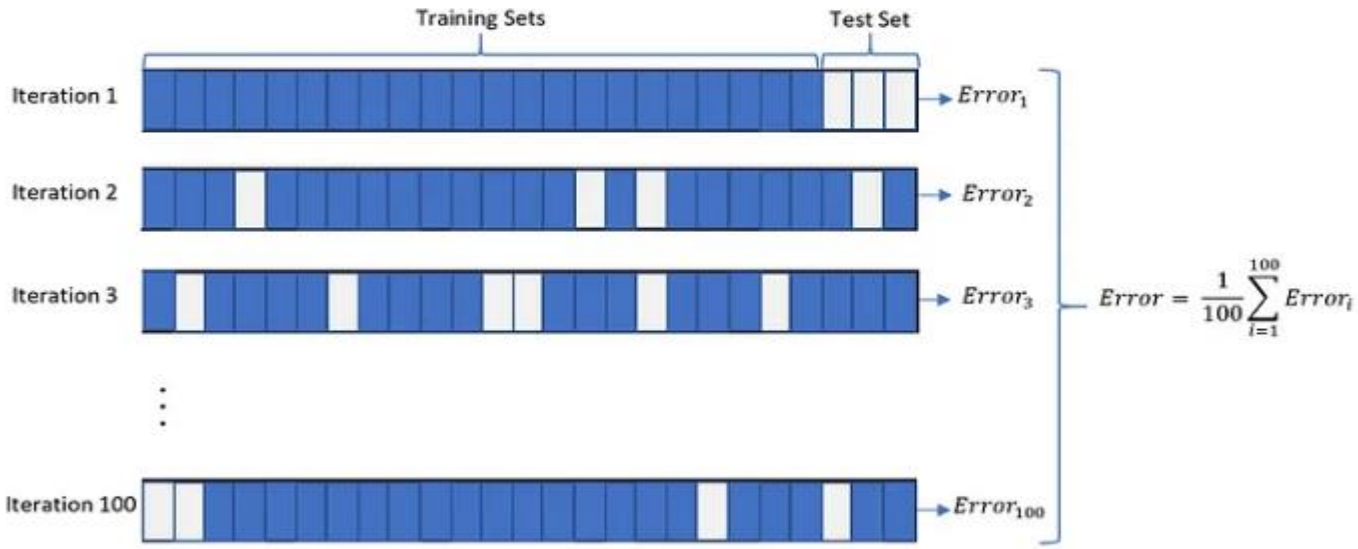

Figure 63 : An example of Regularisation & Cross validation

As i've implemented my dataset by various supervised learning techniques like Logistic Regression, Neural Network, SVM and KNN, i also have the possibility to implement it by Regularisation & Cross validation with the same steps that i've used in the other implementations, each step has its special code as the following codes below :

Step 1 : Importing libraries :

```
#The import of various libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

Figure 64 : Code of importing libraries used to implement by Regularisation & Cross validation

Step 2 : Loading the data :

```
#Load the data
data = pd.read_excel('Immunotherapy.xlsx')
```

Figure 65 : Code of loading my data

Step 3 : Separation and spletting :

```
#Separate the features (X) and the target variable (y)
X = data.drop('Result_of_Treatment', axis=1)
y = data['Result_of_Treatment']
```

Figure 66 : Code of separation

```
#Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 67 : Code of splitting

Step 4 : Feature Scaling :

```
#Perform feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Figure 68 : Code of Feature Scaling

Step 5 : Regularisation & Cross Validation :

```
#Regularisation by normalising the data then allowing cross validation
param_grid = {'C': [0.1, 1, 10]}
logreg = LogisticRegression(penalty='l2', solver='liblinear')
grid_search = GridSearchCV(logreg, param_grid, cv=5)
grid_search.fit(X_train_scaled, y_train)
```

```
GridSearchCV(cv=5, estimator=LogisticRegression(solver='liblinear'),
             param_grid={'C': [0.1, 1, 10]})
```

Figure 69 : Code and result of Regularisation & Cross Validation

Step 6 : Hyperparameter & Accuracy :

```
#Print the best hyperparameters and the corresponding accuracy
print("Best Hyperparameters: ", grid_search.best_params_)
print("Best Accuracy: ", grid_search.best_score_)
```

```
Best Hyperparameters:  {'C': 1}
Best Accuracy:  0.8209523809523809
```

Figure 70 : Code and result of Hyperparameter & Accuracy

Step 7 : Evaluate the model by testing accuracy :

```
#Evaluate the model on the test set
best_model = grid_search.best_estimator_
accuracy = best_model.score(X_test_scaled, y_test)
print("Test Accuracy: ", accuracy)
```

```
Test Accuracy:  0.8333333333333334
```

Figure 71 : Code and result of evaluating the model by testing accuracy

IV. Conclusion :

In conclusion, immunotherapy has emerged as a groundbreaking approach in cancer treatment, offering new possibilities for improved outcomes and enhanced patient care. The introduction of immunotherapeutic interventions has revolutionized the field by harnessing the power of the immune system to fight against cancer cells. Through personalized treatment strategies, immunotherapy has shown promising results in increasing survival rates, extending periods of remission, and improving the quality of life for patients.

However, challenges remain in the widespread implementation of immunotherapy. Access and affordability issues, variable treatment responses, and the need for further research and biomarker identification are among the obstacles to overcome. Collaboration between healthcare systems, policymakers, researchers, and patient advocacy groups is vital to address these challenges and ensure equitable access to immunotherapy for all patients.

Continued investment in research and development is necessary to advance our understanding of immunotherapy mechanisms, refine treatment protocols, and identify novel therapeutic targets. The integration of immunotherapy with other treatment modalities and the development of combination therapies hold promise for further enhancing treatment efficacy.

Moreover, healthcare infrastructure must be strengthened to support the delivery of immunotherapy, including specialized cancer centers, well-trained healthcare professionals, and comprehensive supportive care services. Patient education, awareness, and empowerment are crucial in facilitating shared decision-making and promoting active patient involvement in their treatment journey.

At the end, immunotherapy represents a paradigm shift in cancer treatment, offering immense potential for improved outcomes and personalized care. By addressing the challenges, expanding access, and fostering collaboration, we can overcome barriers and optimize the benefits of immunotherapy, ultimately improving the lives of cancer patients worldwide. The future of immunotherapy holds great promise, and ongoing advancements in this field will continue to reshape the landscape of cancer care.