Breast Cancer Classification (Mobile Application)

Submitted By: -

-Mahmoud Mohamed Rady. **-Mohamed Saeed Koheya.**

-Sara Yasser Mahmoud. **-Anan Adel Noureldeen**

-Shams Kamel Bedir.

Bioinformatics department - faculty of Computers and Information - Menofia university.

Under Supervision: -

Dr. Ahmed Mohamed Kafafy

Contents: -

1: Introduction ……

2: Abstract ……

3: Breast Cancer ……

3.1: Introduction, Definitions …… (7)

3.2: Genetic makeup …… (16)

3.3: Risk Factors …… (19)

3.4: Screening and early detection …… (20)

3.5: Percentage of occurring …… (24)

3.6: Different types …… (25)

4: AI and Machine learning and Deep learning ……

4.1: Introduction of AI …… (30)

4.1: Machine learning …… (31)

4.2: Deep learning …… (32)

4.3: Neural Networks …… (33)

4.4: Machine learning algorithms to know …… (36)

5: The data used and Model with API ……

5.1: Data collection …… (52)

5.2: Data preprocessing …… (53)

5.3: Features selection …… (56)

5.4: Data visualization …… (60)

5.5: The model and the classifier …… (67)

5.6: Saving the model …… (70)

5.7: Creating the API and upload the model …… (71)

6: Flutter ……

6.1: Introduction …… (75)

6.2: Navigation & Routing …… (106)

6.3: Data & Backend …… (108)

6.4: Accessibility & Localization …… (110)

6.5: Packages & Plugins …… (114)

6.6: Testing & Debugging …… (116)

6.7: Performance & Optimization …… (122)

6.8: Deployment …… (124)

6.9: Firebase …… (128)

7: The Application ……

7.1: Sign in …… (134)

7.2: Sign up …… (135)

7.3: Home page …… (140)

7.4: Morphological Features …… (141)

7.5: Shape Features, Result …… (143)

7.6: Testing …… (148)

8: References …… (150)

Introduction

With the widespread rapid development of computers and wireless communication, mobile applications are fast growing also as it's very easy and fast for using.

As of the third quarter of 2019, there were 2.47 million apps on Android's app store and 1.8 million on Apple's app store. Another data suggests that there are more than 12 million registered mobile app developers globally.

So, we decided to build a mobile application for many reasons:

Convenience: Mobile applications bring convenience to users by providing access to various services and information directly on their smartphones or tablets. They eliminate the need to visit a specific website or perform tasks on a desktop computer, allowing users to perform functions on the go.

Accessibility: Mobile apps offer enhanced accessibility compared to websites. They can be easily accessed with a single tap on the app icon, without the need to open a browser and type in a URL. This makes them quicker and more convenient to use, especially for tasks that require frequent access.

Personalization: Mobile apps can personalize the user experience based on preferences, behavior, and location. They can provide tailored content, recommendations, and notifications, making the app more relevant and engaging for each user.

Offline Access: Some mobile applications allow users to access certain features and content even without an internet connection. This is particularly useful when traveling or in areas with limited connectivity.

Device Integration: Mobile apps can integrate with various features and capabilities of a smartphone, such as the camera, GPS, accelerometer, contacts, and more. This integration enables a wide range of functionalities, including photo editing, location-based services, fitness tracking, and social interactions.

Push Notifications: Mobile apps can send push notifications to users, alerting them about updates, new features, promotions, or relevant information. This helps in keeping users engaged and informed about the app and its offerings.

Enhanced User Experience: Mobile apps can provide a more optimized and user-friendly experience compared to mobile websites. They are designed specifically for mobile devices, taking advantage of touch gestures, screen sizes, and other mobile-specific features, resulting in a smoother and more intuitive user experience.

Security: Mobile apps can incorporate security measures such as encryption, secure login, and data protection to ensure the privacy and safety of user information.

Overall, mobile applications offer a range of benefits that enhance user experience, provide convenience, and enable access to services and information anytime, anywhere.

And also we decided to make a mobile application for classifying the breast cancer for some reasons like breast cancer is one of the most common types of cancer worldwide, affecting both women and, to a lesser extent, men. It is the most prevalent cancer among women globally. However, it's important to note that the incidence and prevalence of breast cancer can vary between countries and populations.

According to statistics from the World Health Organization (WHO), breast cancer accounts for approximately 11.7% of all cancer cases worldwide. In 2020, there were an estimated 2.3 million new cases of breast cancer reported globally.

The risk of developing breast cancer increases with age, and the majority of cases occur in women over the age of 50. However, it's essential to recognize that breast cancer can also affect younger women, although at a lower rate.

Early detection through regular screening, such as mammograms, and awareness of the signs and symptoms of breast cancer are crucial in improving outcomes and survival rates. It's recommended that individuals, especially women, perform regular breast self-examinations and seek medical attention if they notice any changes or abnormalities in their breasts.

Abstract

In our project we decided to build a mobile application that can classify breast cancer for two classes Benign or Malignant and we mentioned many information about the breast cancer and many Tips for any one need to know about this dangerous cancer.

By using our application you can classify the breast cancer by knowing some numbers from your doctor or your analysis and inter them in the application and know the result and this numbers are "radius_mean" , "texture_mean" , "perimeter_mean" , "area_mean" , "smoothness_mean" , "compactness_mean" , "concavity_mean" , "concave points_mean" , "symmetry_mean" , "fractal_dimension_mean" , and also you can see some advises that could help you.

Finally we aim to make it easier for anyone to know about his or her health without going for many doctors and in a short time.

Breast cancer

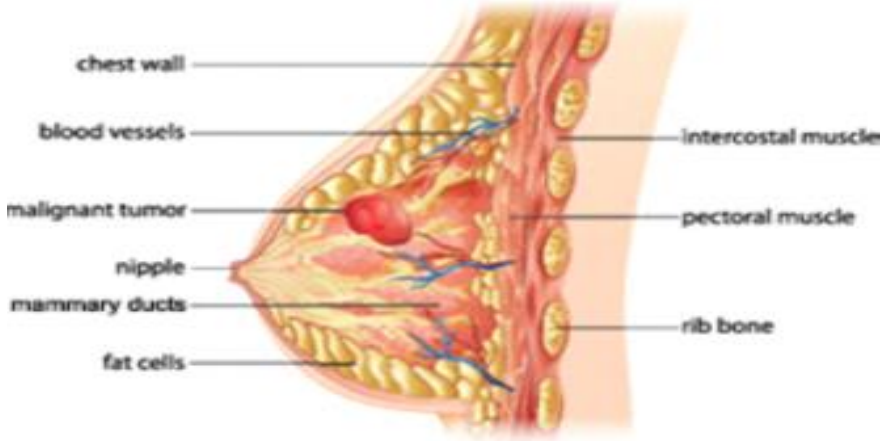
There are many definitions for the breast cancer:

Breast cancer is a type of cancer that develops in the breast tissue when abnormal cells grow uncontrollably.

Breast cancer is a disease that affects the cells in the breast, causing them to grow and divide at an abnormal rate, often forming a lump or tumor.

Breast cancer is a malignant tumor that originates in the breast tissue and can spread to other parts of the body if left untreated.

Breast cancer is a complex disease that can vary in its presentation, severity, and treatment options, and is influenced by a variety of genetic, lifestyle, and environmental factors.



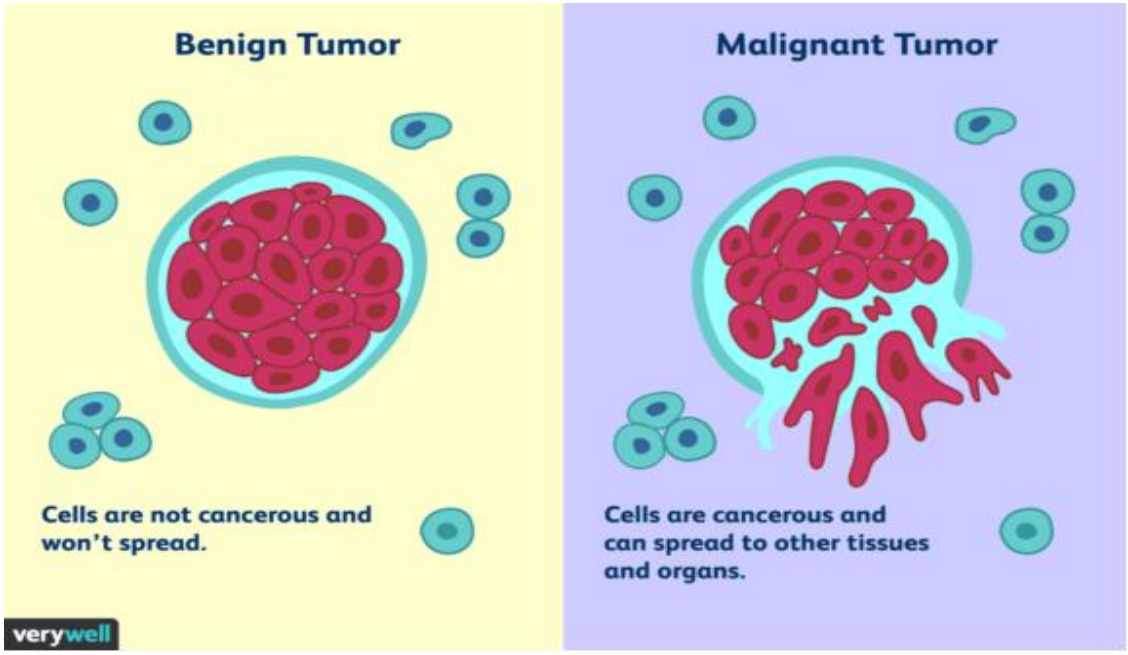
Breast cancer is one of the most common forms of cancer among women worldwide, and in some cases, it can also affect men.

Breast cancer can manifest in different forms, including invasive ductal carcinoma, invasive lobular carcinoma, triple-negative breast cancer, and HER2-positive breast cancer, each with different characteristics and treatment options.

Tumor

What is tumor?

A tumor is an abnormal growth of cells that can occur in any part of the body. Tumors can be benign or malignant, and they can develop in different types of tissue, such as bone, muscle, or organ tissue. In this article, we will provide an overview of tumors, including their causes, types, diagnosis, treatment, and prevention



What are the reasons for the tumor?

The exact causes of tumors are not fully understood, but there are certain risk factors that have been

identified. Some tumors are caused by genetic mutations, while others may be caused by exposure to environmental_toxins or radiation. Lifestyle factors, such as smoking, alcohol consumption, and a poor diet, can also increase the risk of developing certain types of tumors.

About environment toxins

Environmental toxins are substances that can be harmful to human health when they are present in the environment. These toxins can be found in air, water, soil, and food, and they can come from a variety of sources, including industrial and agricultural processes, waste disposal, and consumer products.

Exposure to environmental toxins can have a range of negative health effects, including cancer, respiratory disease, reproductive problems, and developmental delays. The effects of exposure can vary depending on the type and amount of toxin, as well as the age, health status, and genetic makeup of the individual.

Some common environmental toxins include:

- **Lead: Lead is a toxic metal that can be found in paint, water pipes, and soil. Exposure to lead can cause developmental delays, behavioral problems, and anemia.**
- **Mercury: Mercury is a toxic metal that can be found in some types of fish, as well as in certain types of industrial processes. Exposure to mercury can cause neurological damage, particularly in infants and young children.**

- **Pesticides: Pesticides are chemicals used to kill pests, such as insects and rodents. Exposure to pesticides can cause cancer, reproductive problems, and neurological damage.**
- **Air pollution: Air pollution, such as particulate matter and ozone, can be emitted from vehicles, power plants, and other sources. Exposure to air pollution can cause respiratory problems, such as asthma and lung cancer.**
- **Plastics: Many plastics contain chemicals, such as bisphenol A (BPA) and phthalates, which can leach into food and water. Exposure to these chemicals can disrupt hormone function and increase the risk of cancer.**

To reduce exposure to environmental toxins, individuals can take several good steps, including:

- **Eating a healthy diet: Eating a diet rich in fruits, vegetables, and whole grains can help reduce the risk of exposure to environmental toxins.**
- **Drinking clean water: Drinking water from a safe source, such as a municipal water supply or a filtered**

well, can help reduce the risk of exposure to environmental toxins.

- **Reducing pesticide exposure: Choosing organic produce and avoiding the use of pesticides in the home and garden can help reduce exposure to these chemicals.**
- **Using safe consumer products: Choosing products that are free of harmful chemicals, such as BPA and phthalates, can help reduce the risk of exposure.**
- **Reducing air pollution: Using public transportation, walking, or biking instead of driving can help reduce exposure to air pollution, as can supporting policies that reduce emissions from industrial sources.**

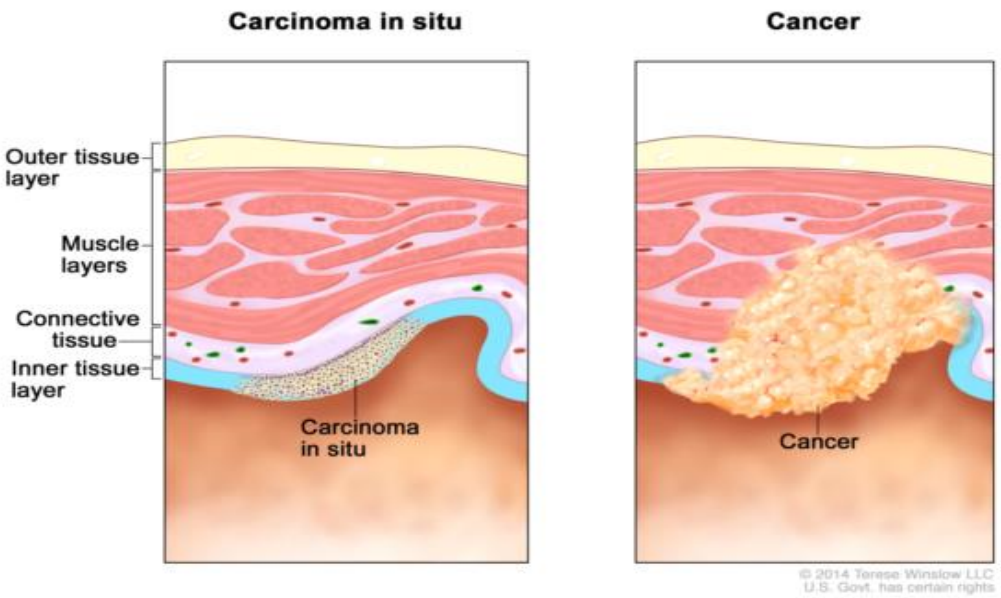
In conclusion, exposure to environmental toxins can have serious health effects, but there are steps individuals can take to reduce their risk. By staying informed about the sources of environmental toxins and taking proactive steps to protect themselves and their families, individuals can help ensure a healthier future.

Types of Tumors

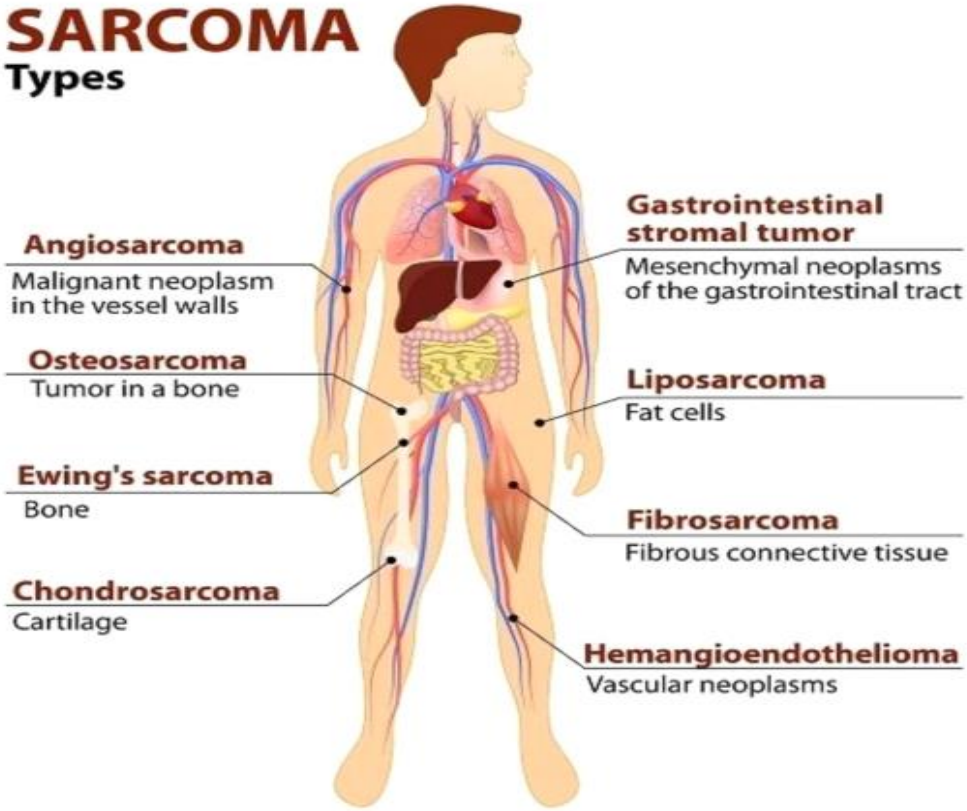
Tumors can be classified as either benign or malignant. Benign tumors are non-cancerous and do not spread to other parts of the body. They are typically slow-growing and can often be removed with surgery. Malignant tumors, on the other hand, are cancerous

and can spread to other parts of the body if left untreated. They can be more aggressive and require more complex treatment options, such as chemotherapy, radiation therapy, or surgery.
Tumors can also be classified based on the type of tissue in which they develop. Some common types of tumors include:

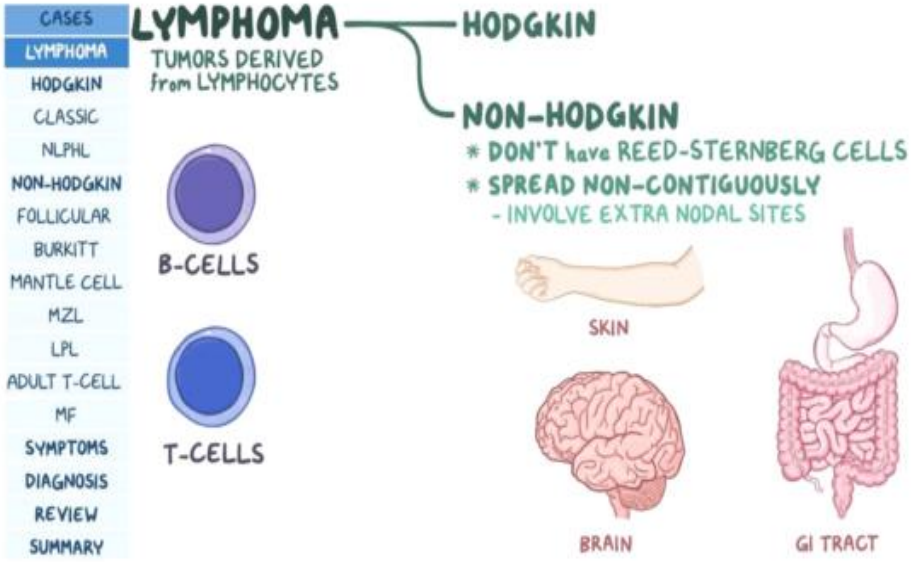
- **Carcinomas: These tumors develop in the epithelial tissue, which covers the surface of organs and glands. Examples include lung cancer and breast cancer.**



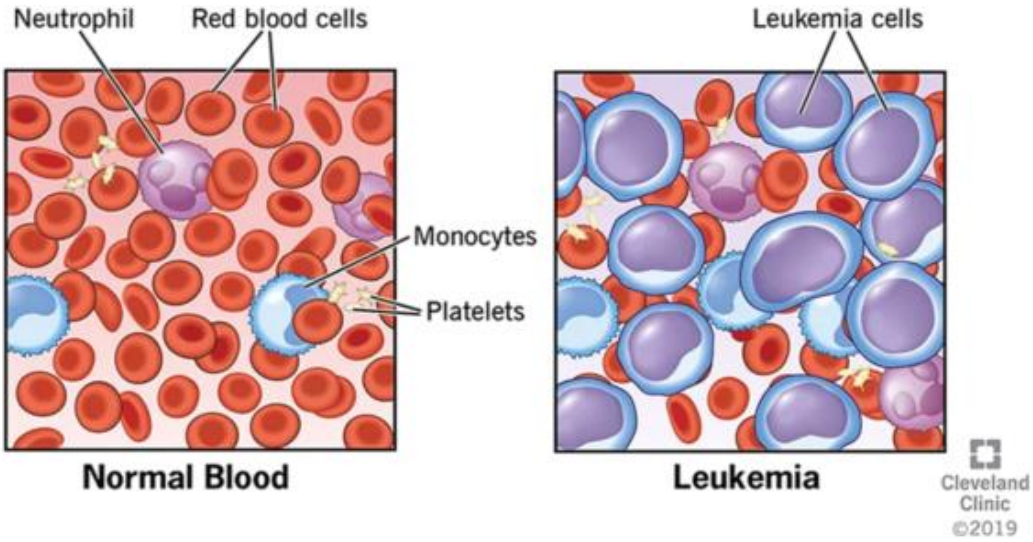
- **Sarcomas: These tumors develop in the connective tissue, such as bone, muscle, or cartilage. Examples include osteosarcoma and leiomyosarcoma.**



- **Lymphomas: These tumors develop in the lymphatic system, which is part of the immune system. Examples include Hodgkin's lymphoma and non-Hodgkin's lymphoma.**
-



- **Leukemias: These tumors develop in the blood-forming cells and can affect the bone marrow and blood. Examples include acute lymphoblastic leukemia and chronic myeloid leukemia.**



Diagnosis of Tumors

Tumors are typically diagnosed through a combination of physical exams, imaging tests, and biopsies.
A physical exam involves a healthcare provider examining the affected area for any signs of a tumor. Imaging tests, such as X-rays, CT scans, and MRI scans, can be used to visualize the tumor and determine its size and location. A biopsy involves removing a small amount of tissue for examination under a microscope to determine whether the tumor is benign or malignant.

Treatment of Tumors

The treatment options for tumors depend on the type, stage, and location of the tumor, as well as the patient's overall health and preferences. Treatment

options for benign tumors may include watchful waiting, medication, or surgery to remove the tumor. Treatment options for malignant tumors may include chemotherapy, radiation therapy, surgery, or a combination of these treatments. In some cases, clinical trials may be available for patients with advanced or difficult-to-treat tumors.

Prevention of Tumors

While not all tumors can be prevented, there are certain steps that individuals can take to reduce their risk of developing certain types of tumors. These include:

- **Maintaining a healthy lifestyle: Eating a balanced diet, getting regular exercise, avoiding smoking, and limiting alcohol consumption can all help reduce the risk of developing certain types of tumors.**
- **Protecting against environmental toxins: Avoiding exposure to environmental toxins, such as pesticides and radiation, can also help reduce the risk of developing certain types of tumors.**
- **Getting regular screenings: Regular cancer screenings, such as mammograms and colonoscopies, can help detect tumors early when they are easier to treat.**

Tumors are a common and complex health issue that can have significant physical and emotional impacts on patients and their families. While the causes of tumors are not fully understood, there are certain risk factors that have been identified. Early detection and treatment can improve the chances of survival, and there are certain steps that individuals can take to reduce their risk of developing certain types of tumors. By staying informed about the disease and taking proactive steps to prevent it, individuals can help protect their health and well-being.

genetic makeup

Genetic makeup refers to the unique set of genes that an individual inherits from their parents. Genes are the basic units of heredity, and they contain the

instructions for building and maintaining the body's cells, tissues, and organs. Genetic makeup can influence a wide range of traits, including physical characteristics, such as eye color and height, as well as traits related to health and disease.

There are two types of genes that make up an individual's genetic makeup:

- **Dominant and recessive**
- **Dominant genes are those that are expressed when present, meaning that they will produce a particular trait even if only one copy is present.**
- **Recessive genes, on the other hand, are only expressed when two copies are present.**

Genetic makeup can also be influenced by variations in gene expression, which refers to the way in which genes are turned on or off in response to environmental factors. For example, stress, nutrition, and exposure to toxins can all influence gene expression and impact an individual's health and well-being.

Genetic makeup can influence a wide range of health outcomes, including the risk of developing certain diseases. Some genetic conditions, such as sickle cell anemia or cystic fibrosis, are caused by mutations in a single gene and are typically inherited in a predictable pattern. Other conditions, such as heart disease and cancer, are influenced by multiple genes as well as environmental factors and lifestyle choices.

Advances in genetic research

Advances in genetic research have led to the development of genetic testing, which can help identify individuals who are at increased risk of certain diseases based on their genetic makeup.

Genetic testing can also be used to diagnose genetic disorders and guide treatment decisions.

However, it is important to note that genetic makeup is the only factor that influences health outcomes. Environmental factors, such as diet, exercise, and exposure to toxins, can also play a significant role in determining an individual's health status. Additionally, genetic makeup is not necessarily deterministic, meaning that individuals can take steps to reduce their risk of developing certain diseases through healthy lifestyle choices.

In conclusion, genetic makeup refers to the unique set of genes that an individual inherits from their parents and can influence a wide range of traits, including health outcomes. While genetic testing can provide

valuable information about an individual's risk of certain diseases, it is important to remember that genetic makeup is only one factor that influences health, and that healthy lifestyle choices can also have a significant impact on overall well-being.

risk factors

Breast cancer is a complex disease that can be influenced by a variety of factors, including genetic, environmental, and lifestyle factors. While the exact cause of breast cancer is not fully understood, there are several known risk factors that can increase the likelihood of developing the disease.

Some of the most common risk factors for breast cancer include:

- **Age: The risk of breast cancer increases as a person gets older, with most cases diagnosed in women over the age of 50.**
- **Gender: Breast cancer is more common in women than men, although men can also develop the disease.**
- **Family history: Women with a family history of breast cancer, particularly those with a mother, sister, or daughter who has been diagnosed with the disease, are at increased risk.**
- **Genetic mutations: Inherited mutations in the BRCA1 and BRCA2 genes can increase the risk of breast cancer, as can other genetic mutations.**

- **Hormonal factors: Excessive exposure to estrogen, a hormone that promotes the growth of breast cells, can increase the risk of breast cancer. Factors that can increase estrogen exposure include early onset of menstruation, late onset of menopause, and using hormonal replacement therapy.**
- **Lifestyle factors: Certain lifestyle factors, such as smoking, alcohol consumption, and a poor diet, can increase the risk of developing breast cancer.**
- **Environmental factors: Exposure to environmental toxins, such as pesticides and radiation, can also increase the risk of breast cancer.**

It is important to note that having one or more of these risk factors does not necessarily mean that an individual will develop breast cancer. Additionally, some people may develop breast cancer without any known risk factors. Regular breast cancer screening and early detection can improve the chances of successful treatment and recovery, regardless of risk factors.

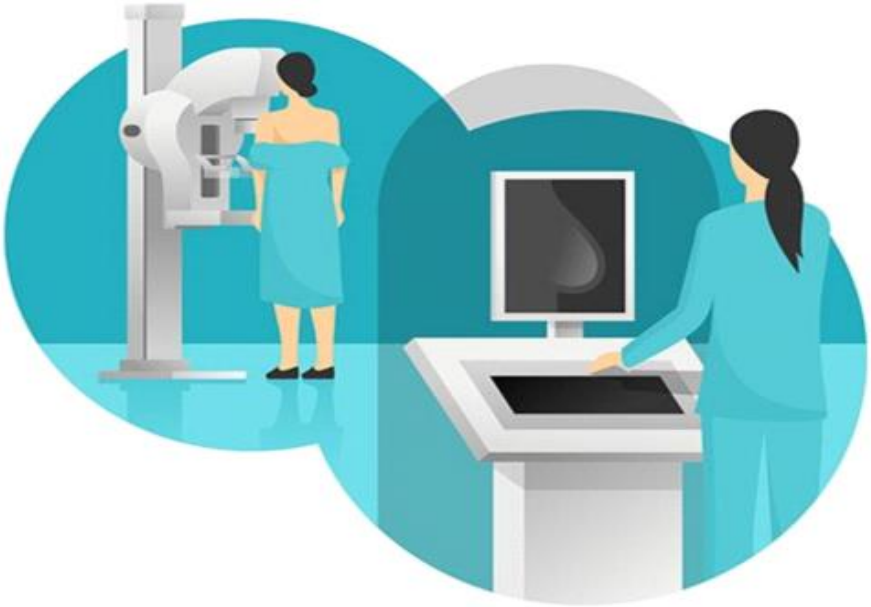
Breast cancer screening and early detection

Breast cancer screening refers to the process of checking for signs of breast cancer in women who have no symptoms of the disease. The goal of breast cancer screening is to detect cancer in its early stages, when it

is more treatable and has a better chance of being cured.

There are several different methods of breast cancer screening, including:

- **Mammography: Mammography is the most common screening tool for breast cancer. It uses X-rays to create images of the breast tissue and can detect small tumors that may not be felt during a physical exam.**



- **Clinical breast exam: A clinical breast exam is a physical exam performed by a healthcare provider to check for lumps or other abnormalities in the breast tissue.**
- **Breast self-exam: A breast self-exam is a technique that women can use to check their own breasts for lumps or other changes.**
- **Magnetic resonance imaging (MRI): MRI uses magnetic fields and radio waves to create detailed images of the breast tissue. It may be used in**

addition to mammography for women who are at high risk of developing breast cancer.

The American Cancer Society recommends that women with an average risk of breast cancer begin screening at age 45, with yearly mammograms recommended between the ages of 45 and 54. After age 55, women may switch to screening every two years, or continue with yearly screening if they choose. Women at high risk of breast cancer may need to begin screening at an earlier age or have more frequent screening.

It is important to note that breast cancer screening has both benefits and limitations. While screening can detect cancer in its early stages and improve treatment outcomes, it can also lead to false positives, which can cause anxiety and unnecessary testing. Additionally, not all breast cancers can be detected through screening, and some cancers may not require treatment.

Women should talk to their healthcare provider about their individual risk of breast cancer and the most appropriate screening schedule for their needs. Regular breast cancer screening, along with healthy lifestyle choices, can help improve the chances of detecting and treating breast cancer early.

Early detection refers to the process of detecting a disease in its earliest stages before symptoms develop or when symptoms are mild. Early detection is

important because it can improve treatment outcomes and increase the chances of a successful recovery.

In the context of cancer, early detection can involve screening tests, such as mammography for breast cancer or colonoscopy for colorectal cancer, that are used to detect cancer before symptoms develop. Early detection can also involve recognizing the signs and symptoms of cancer and seeking medical attention as soon as possible.

The benefits of early detection for cancer include:

- **Better treatment outcomes: When cancer is detected early, it is often easier to treat and has a better chance of being cured.**
- **Less invasive treatment: When cancer is detected early, it may be possible to treat it with less invasive procedures, such as surgery or radiation therapy, rather than more extensive treatments like chemotherapy.**
- **Improved quality of life: Early detection and treatment of cancer can help prevent the spread of the disease and improve quality of life for the patient.**
- **Lower healthcare costs: Early detection can reduce the need for more extensive and costly treatments, which can lower overall healthcare costs.**

In addition to cancer, early detection can also be important for other health conditions, such as heart disease and diabetes. Regular check-ups with a

healthcare provider, along with routine health screenings and tests, can help detect these conditions early and improve treatment outcomes.
It is important to note that early detection does not guarantee a cure, and that some cancers and other health conditions may not be detectable through screening or may not have noticeable symptoms in their early stages. However, early detection can still be an important tool for reducing the impact of disease on individuals and improving overall health outcomes.
In conclusion, early detection involves detecting a disease in its earliest stages, before symptoms develop or when symptoms are mild. Early detection can improve treatment outcomes, reduce healthcare costs, and improve quality of life for patients. Regular health check-ups and screenings can help detect diseases early and improve overall health outcomes.

Percentage of breast cancer occurring

Breast cancer is one of the most common cancers among women worldwide. The exact percentage of breast cancer cases varies depending on factors such as age, race, and geographic location. However, according to the World Health Organization (WHO), breast cancer is estimated to be responsible for 11.7% of all cancer cases and 6.9% of all cancer deaths among women globally.
In the United States, breast cancer is the second most common cancer among women, after skin cancer. The

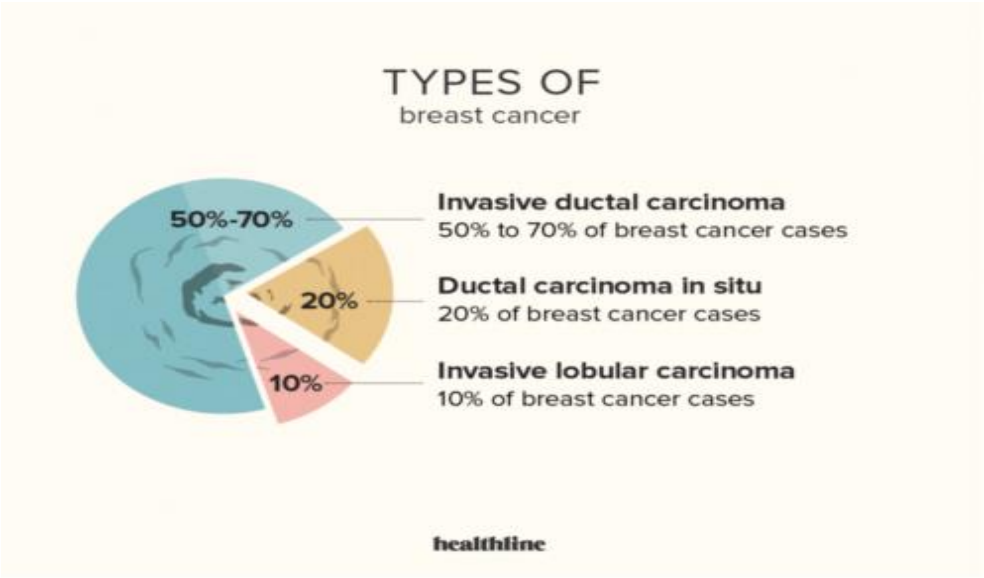
American Cancer Society estimates that about 1 in 8 women in the US will develop invasive breast cancer at some point in their lifetime. In 2021, it is estimated that there will be about 281,550 new cases of invasive breast cancer diagnosed in women in the US, as well as about 49,290 new cases of non-invasive (in situ) breast cancer.

Breast cancer can also occur in men, although it is much less common. According to the American Cancer Society, less than 1% of all breast cancer cases occur in men.

It is important to note that while breast cancer is relatively common, not all breast lumps or changes are cancerous. Other conditions, such as benign breast tumors or cysts, can also cause lumps or changes in the breast tissue. Women who notice any changes in their breasts, such as a lump or thickening, nipple discharge, or skin changes, should seek medical attention to determine the cause of their symptoms. Regular breast cancer screening, as recommended by healthcare providers, can also help detect breast cancer in its early stages when it is more treatable.

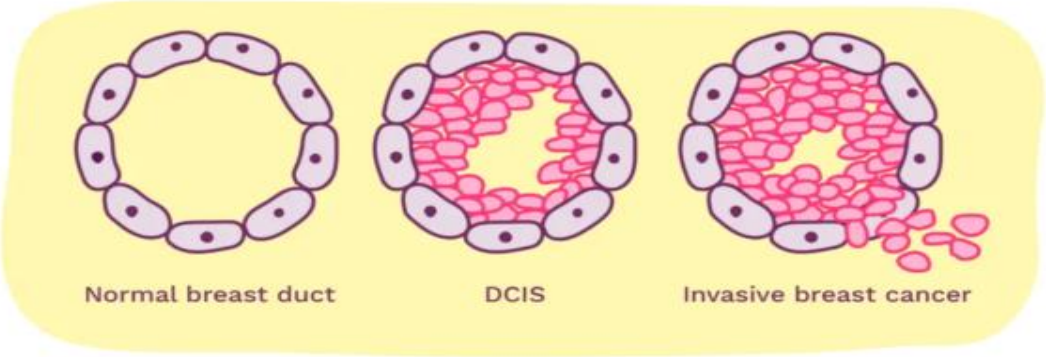
Types of breast cancer

There are several different types of breast cancer, which are classified based on the type of cells that are involved in the cancer.

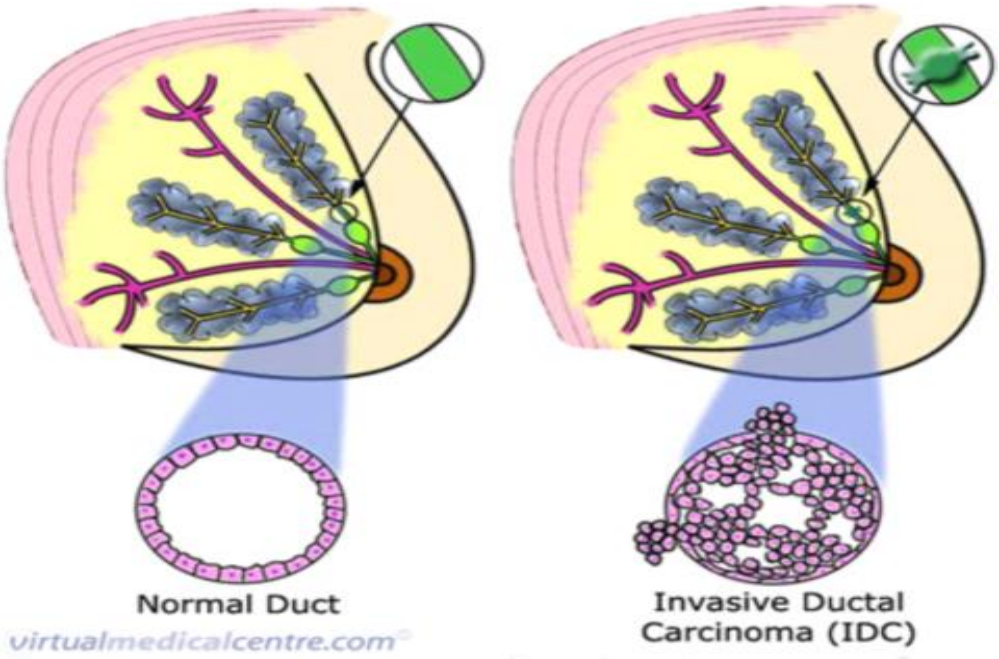


The most common types of breast cancer include:

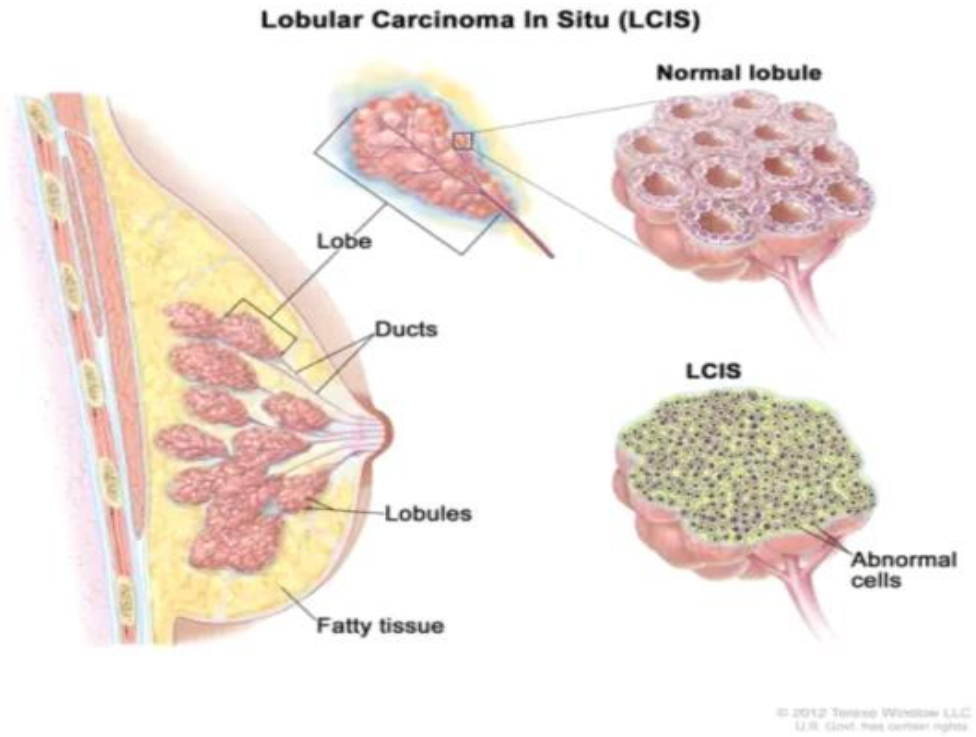
- **Ductal carcinoma in situ (DCIS): DCIS is a non-invasive cancer that occurs in the milk ducts of the breast. It is sometimes referred to as "stage 0" breast cancer because it has not yet spread outside of the milk ducts. DCIS is usually detected through mammography and is highly treatable.**



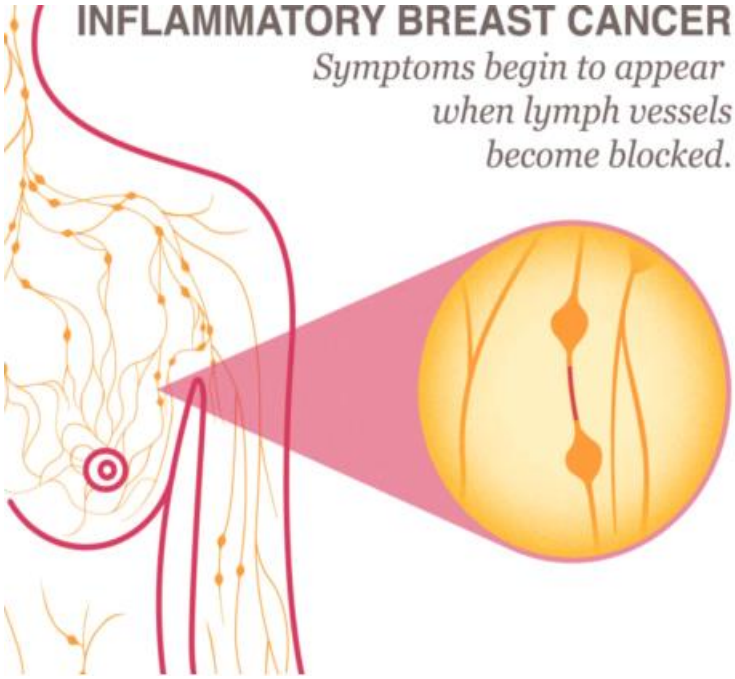
- **Invasive ductal carcinoma (IDC): IDC is the most common type of breast cancer, accounting for about 80% of all breast cancer cases. It occurs when cancer cells invade the surrounding breast tissue outside of the milk ducts. IDC can spread to other parts of the body if left untreated.**



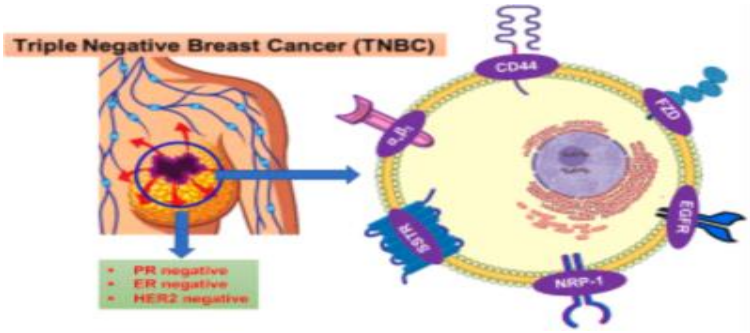
- **Invasive lobular carcinoma (ILC): ILC is a type of breast cancer that begins in the milk-producing glands (lobules) of the breast and can spread to other parts of the body if left untreated. It accounts for about 10-15% of all breast cancer cases.**



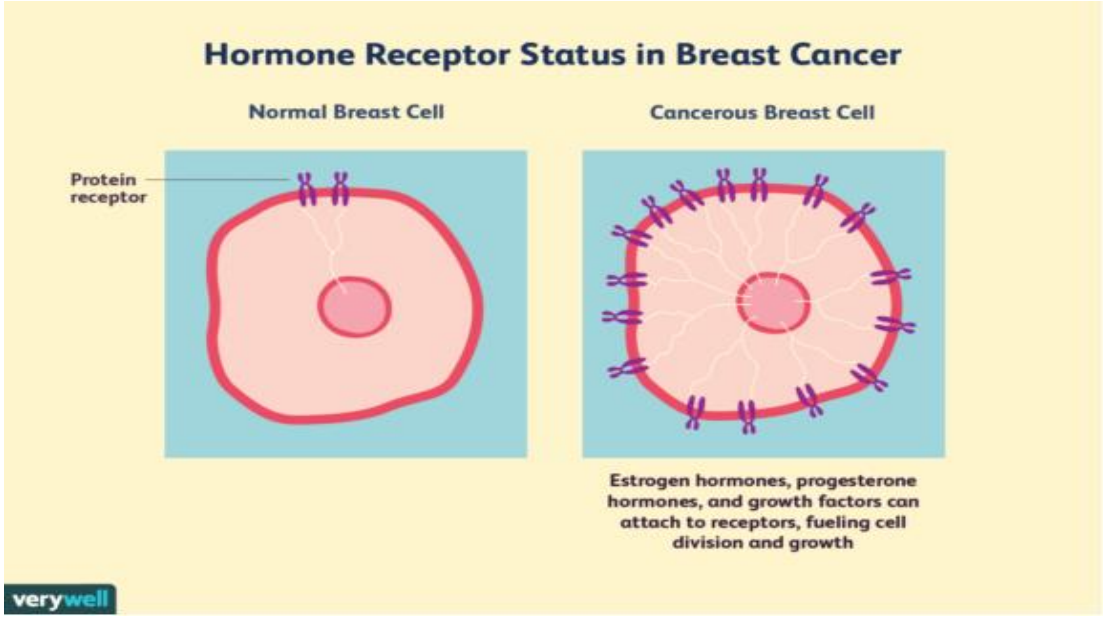
- **Inflammatory breast cancer (IBC): IBC is a rare but aggressive form of breast cancer that occurs when cancer cells block the lymphatic vessels in the skin of the breast, causing the breast to become red, swollen, and warm to the touch. IBC is often misdiagnosed as an infection and requires prompt medical attention.**



- **Triple-negative breast cancer (TNBC): TNBC is a type of breast cancer that does not have receptors for estrogen, progesterone, or HER2, which are the three proteins that can fuel the growth of breast cancer cells. TNBC is often more aggressive than other types of breast cancer and has limited treatment options.**



- **Hormone receptor-positive breast cancer: This type of breast cancer has receptors for estrogen and/or progesterone, which means that the cancer cells are fueled by these hormones. Hormone receptor-positive breast cancer can be treated with hormone therapy, which blocks the hormones from fueling the growth of cancer cells.**

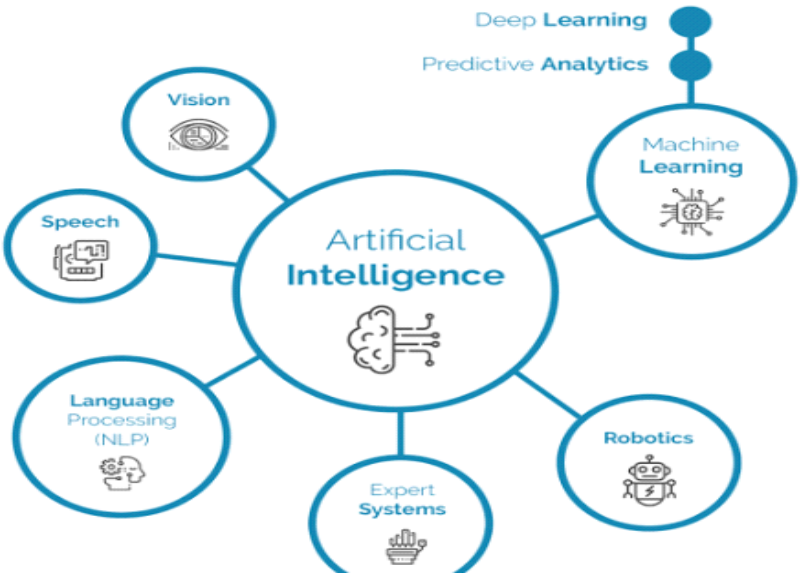


There are also several fewer common types of breast cancer, such as phyllodes tumors and angiosarcoma, which are rare but can be aggressive. The type of breast cancer a person has can impact their treatment options and prognosis, so it is important to receive an accurate diagnosis.

Artificial Intelligence (AI)

Artificial Intelligence is the field of developing computers and robots that are capable of behaving in ways that both mimic and go beyond human capabilities. AI-enabled programs can analyze and contextualize data to provide information or automatically trigger actions without human interference.

Today, artificial intelligence is at the heart of many technologies we use, including smart devices and voice assistants such as Siri on Apple devices. Companies are incorporating techniques such as natural language processing and computer vision — the ability for computers to use human language and interpret images -— to automate tasks, accelerate decision making, and enable customer conversations with chatbots.

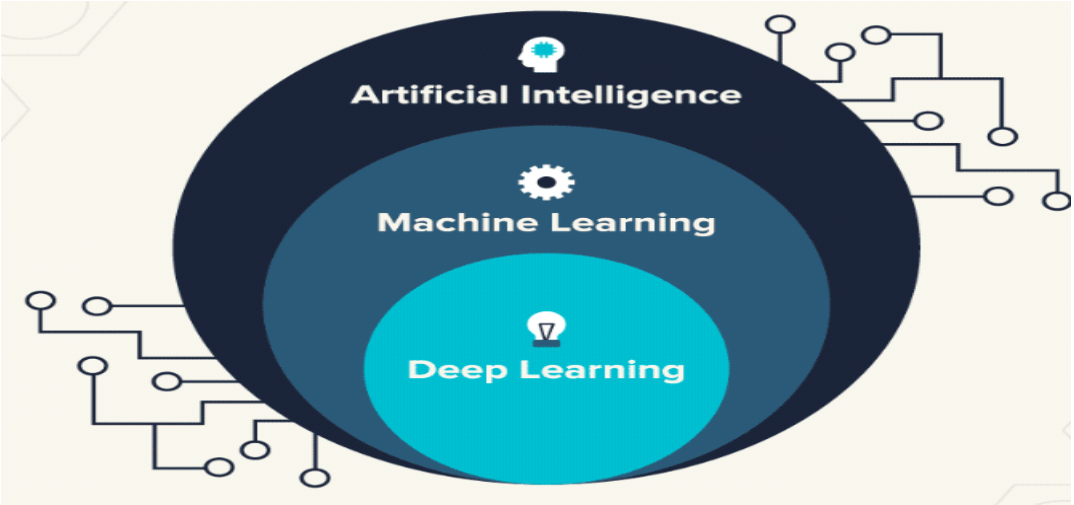


Machine Learning

Machine learning is a pathway to artificial intelligence. This subcategory of AI uses algorithms to automatically learn insights and recognize patterns from data, applying that learning to make increasingly better decisions.

By studying and experimenting with machine learning, programmers test the limits of how much they can improve the perception, cognition, and action of a computer system.

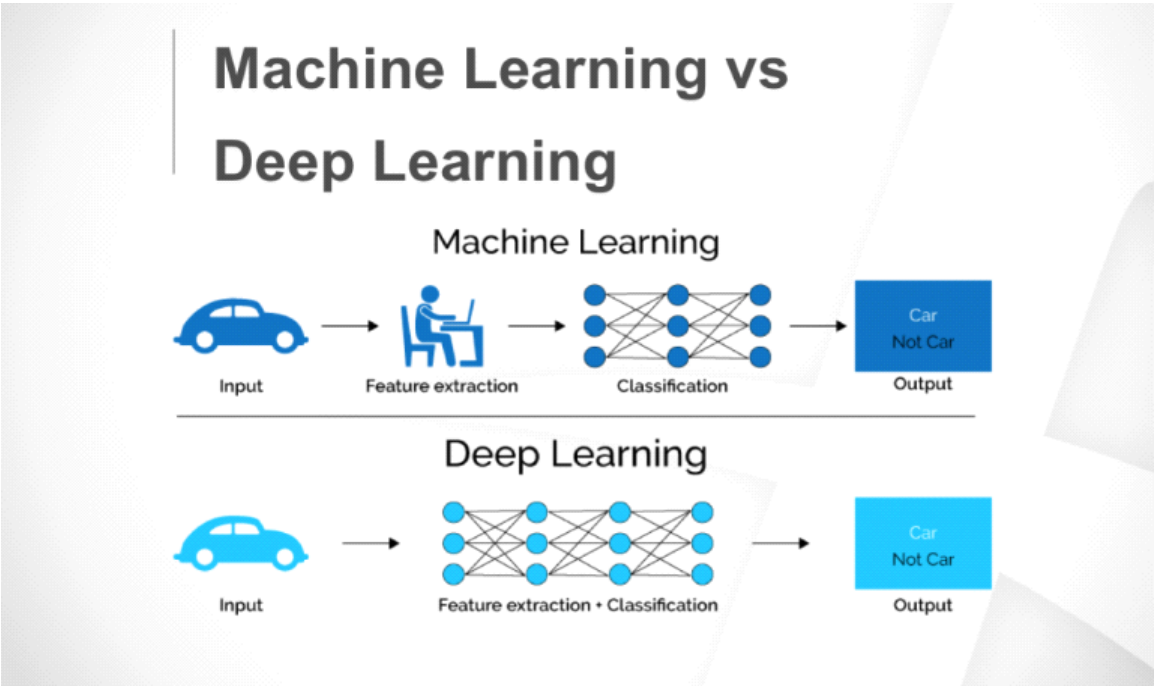
The field of machine learning has been revolutionized by the development of deep learning. Deep learning is a subset of machine learning that uses artificial neural networks to model and solve complex problems. The algorithms used in deep learning have shown remarkable success in a wide range of applications, including image and speech recognition, natural language processing, and autonomous vehicles.



Deep Learning

Deep learning is a type of machine learning that involves the use of neural networks, which are modeled after the human brain. These neural networks consist of layers of interconnected nodes, or neurons, that are capable of processing information and making predictions. Deep learning algorithms are designed to learn from data and improve their performance over time through a process called training.

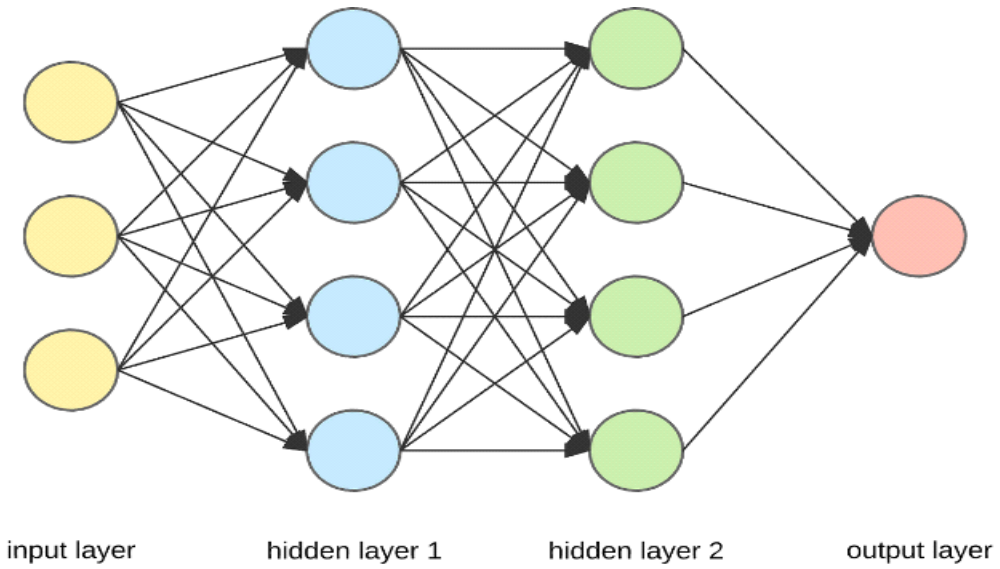
One of the key advantages of deep learning is its ability to handle large and complex datasets. Deep learning algorithms can automatically extract features from raw data, allowing them to learn and generalize patterns that may not be immediately apparent to human observers.



Neural Networks

Neural networks are the building blocks of deep learning. They consist of layers of interconnected nodes, or neurons, that process information and make predictions. Each neuron in a neural network has an associated weight and bias, which are used to calculate the output of the neuron. The output of one layer of neurons is fed as input to the next layer, allowing the network to make increasingly complex predictions.

Training a neural network involves adjusting theweights and biases of the neurons based on the error between the predicted output and the actual output. This process is known as backpropagation, and it involves computing the gradient of the error with respect to each weight and bias in the network. The weights and biases are then updated using an optimization algorithm such as stochastic gradient descent.

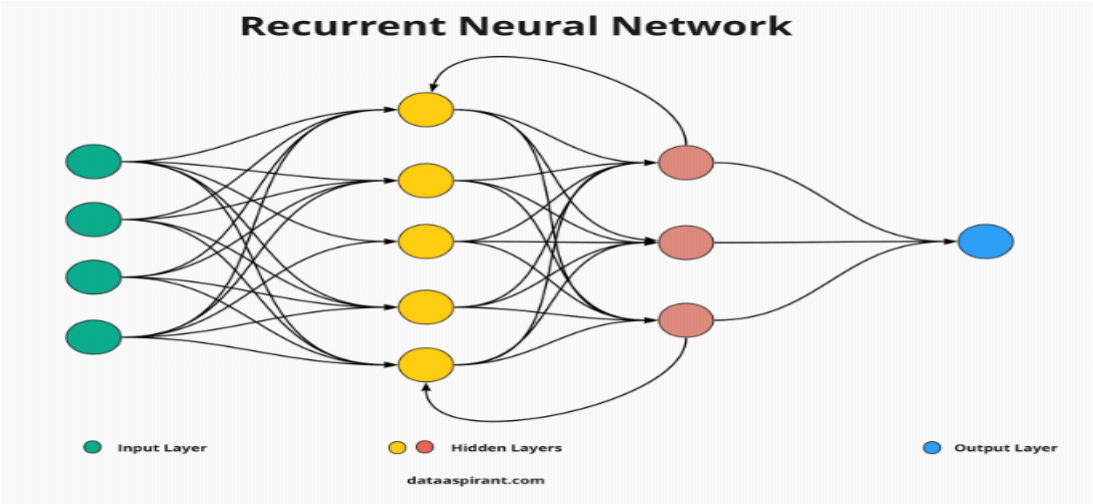


Convolutional Neural Networks

Convolutional neural networks (CNNs) are a type of neural network that are particularly well-suited for image and video processing tasks. They use filters, or kernels, to extract features from the input images. These filters are applied to local regions of the image, and the resulting feature maps are fed as input to the next layer of the network. This process is repeated multiple times, allowing the network to learn increasingly complex features.

Recurrent Neural Networks

Recurrent neural networks (RNNs) are a type of neural network that are well-suited for sequential data, such as natural language processing or time series analysis. RNNs use a feedback loop to allow information to flow from one time step to the next. This allows the network to maintain an internal state that can capture dependencies between inputs at different time steps.



Long Short-Term Memory

Long short-term memory (LSTM) is a type of RNN that is designed to handle long-term dependencies. LSTMs use a memory cell to store information over multiple time steps, and a set of gates to control the flow of information into and out of the memory cell.Random Forest

Random forest is a popular machine learning algorithm that is often used for classification and regression tasks. It is an ensemble learning method that combines multiple decision trees to make predictions. Each decision tree in the random forest is trained on a subset of the training data and a subset of the features.

The random forest algorithm works by randomly selecting a subset of the training data and a subset of the features for each decision tree. This helps to reduce overfitting and improve the generalization performance of the model. Once the decision trees have been trained, the random forest combines their predictions to make a final prediction.

One of the key advantages of random forest is its ability to handle high-dimensional data with many features. It is also robust to missing data and outliers. Random forest can be used for both classification and

regression tasks, and it has been shown to perform well on a wide range of datasets.

machine learning algorithms to know

In simple terms, a machine learning algorithm is like a recipe that allows computers to learn and make predictions from data. Instead of explicitly telling the computer what to do, we provide it with a large amount of data and let it discover patterns, relationships, and insights on its own.

From classification to regression, here are 9 algorithms you need to know in the field of machine learning:

1. Random forest

A random forest algorithm is an ensemble of decision trees used for classification and predictive modeling. Instead of relying on a single decision tree, a random forest combines the predictions from multiple decision trees to make more accurate predictions.

In a random forest, numerous decision tree algorithms (sometimes hundreds or even thousands) are individually trained using different random samples from the training dataset. This sampling method is called "bagging." Each decision tree is trained independently on its respective random sample.

Once trained, the random forest takes the same data and feeds it into each decision tree. Each tree produces a prediction, and the random forest tallies the results. The most common prediction among all the decision trees is then selected as the final prediction for the dataset.

Random forests address a common issue called "overfitting" that can occur with individual decision trees. Overfitting happens when a decision tree becomes too closely aligned with its training data, making it less accurate when presented with new data.

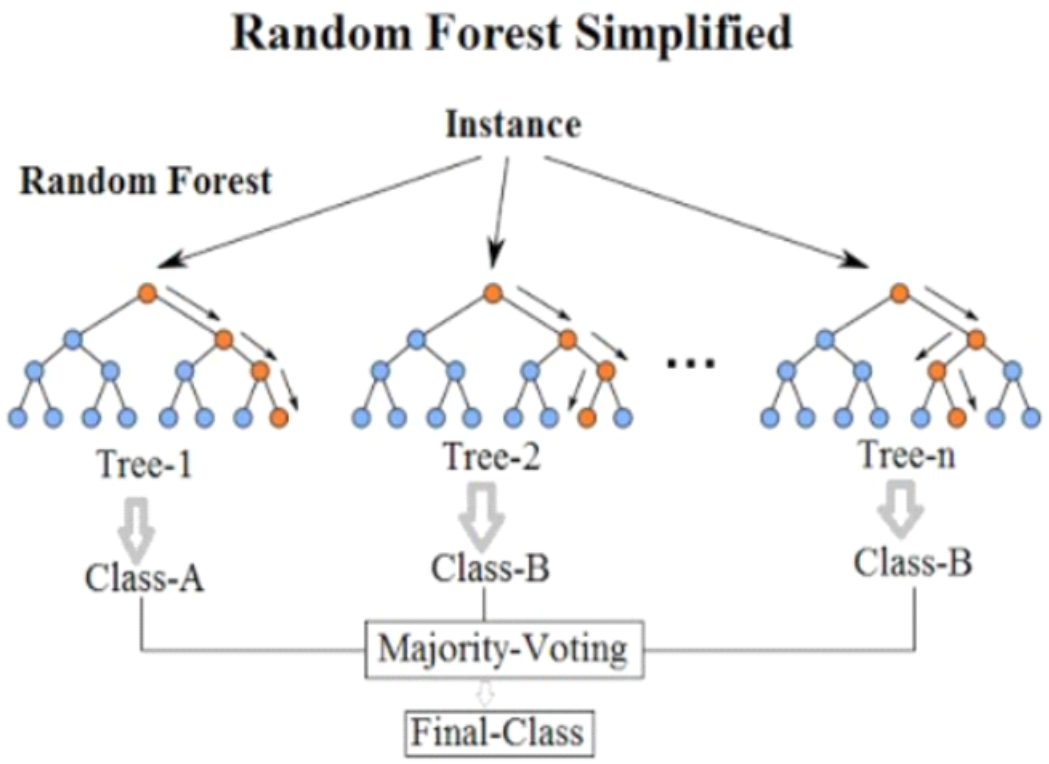


Illustration of Random Forest:

Suppose we have a dataset of housing prices, with features such as the number of bedrooms, square footage, and location. We want to use a random forest to predict the price of a new house based on its features.

First, we randomly select a subset of the training data and a subset of the features to train the first decision tree. The decision tree splits the data based on the selected features, creating subsets of data that are more homogeneous in terms of the target variable (price).

Next, we repeat this process to train multiple decision trees. Each decision tree is trained ona different subset of the data and features, leading to slightly different splits and predictions.

To make a prediction for a new house, we feed its features into each decision tree in the random forest. Each decision tree makes a prediction based on its split and the features, and the final prediction is the average (in regression) or majority vote (in classification) of the predictions from all the decision trees.

RANDOM FOREST ADVANTAGES:

One of the biggest advantages of random forest is its versatility. It can be used for both regression and classification tasks, and it's also easy to view the relative importance it assigns to the input features.

Versatile uses

Easy-to-understand hyperparameters

Classifier doesn't overfit with enough trees

Random forest is also a very handy algorithm because the default hyperparameters it uses often produce a good prediction result. Understanding the hyperparameters is pretty straightforward, and there's also not that many of them.

One of the biggest problems in machine learning is overfitting, but most of the time this won't happen thanks to the random forest classifier. If there are enough trees in the forest, the classifier won't overfit the model.

RANDOM FOREST DISADVANTAGES:

The main limitation of random forest is that a large number of trees can make the algorithm too slow and ineffective for real-time predictions. In general, these

algorithms are fast to train, but quite slow to create predictions once they are trained. A more accurate prediction requires more trees, which results in a slower model. In most real-world applications, the random forest algorithm is fast enough but there can certainly be situations where run-time performance is important and other approaches would be preferred.

Increased accuracy requires more trees slow down model

Can't describe relationships within data

And, of course, random forest is a predictive modeling tool and not a descriptive tool, meaning if you're looking for a description of the relationships in your data, other approaches would be better.

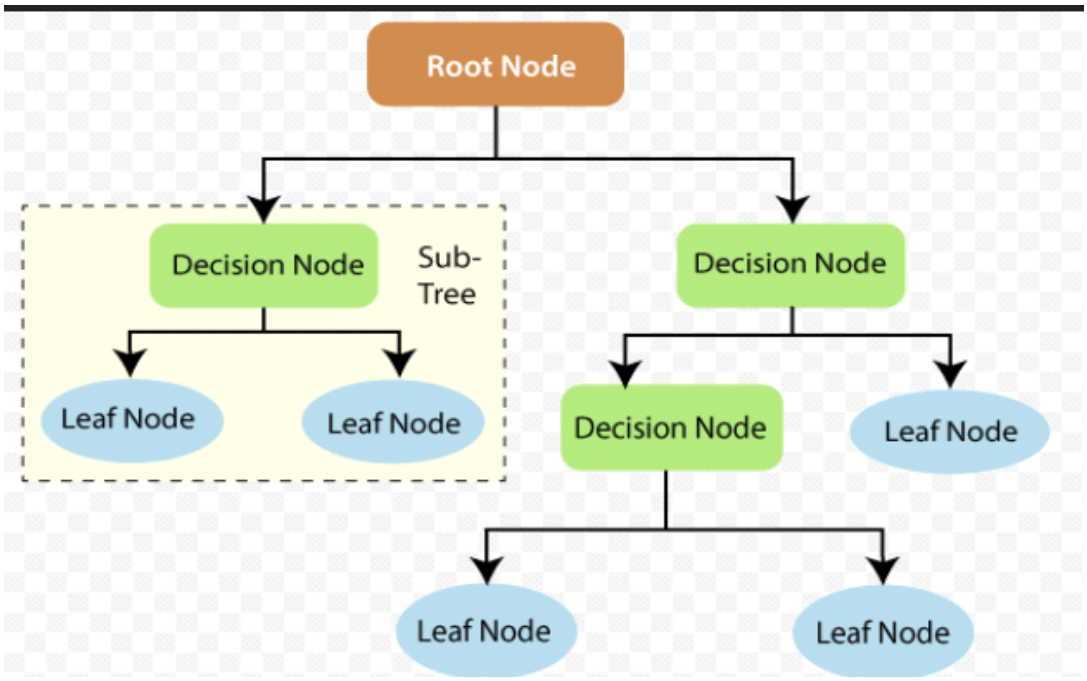
2. Decision tree

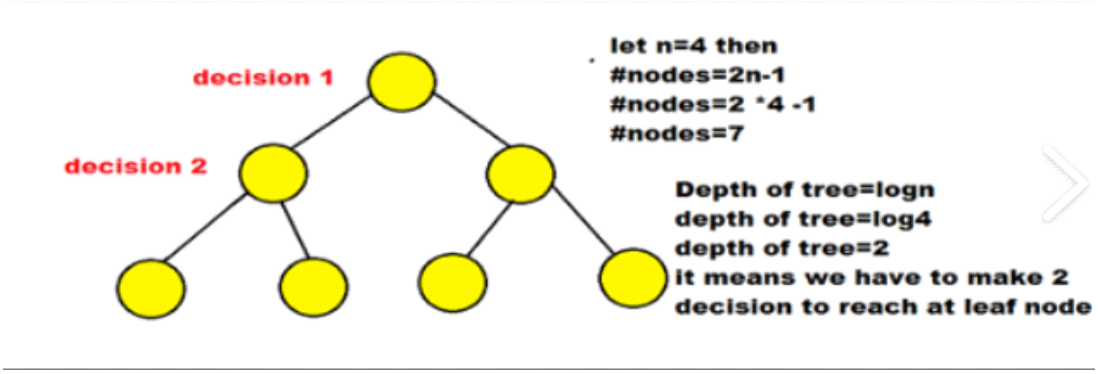
A decision tree is a supervised learning algorithm used for classification and predictive modeling tasks. It resembles a flowchart, starting with a root node that asks a specific question about the data. Based on the answer, the data is directed down different branches to subsequent internal nodes, which ask further questions and guide the data to subsequent branches. This process continues until the data reaches an end

node, also known as a leaf node, where no further branching occurs.

Decision tree algorithms are popular in machine learning because they can handle complex datasets with ease and simplicity. The algorithm's structure makes it straightforward to understand and interpret the decision-making process. By asking a sequence of questions and following the corresponding branches, decision trees enable us to classify or predict outcomes based on the data's characteristics.

This simplicity and interpretability make decision trees valuable for various applications in machine learning, especially when dealing with complex datasets.





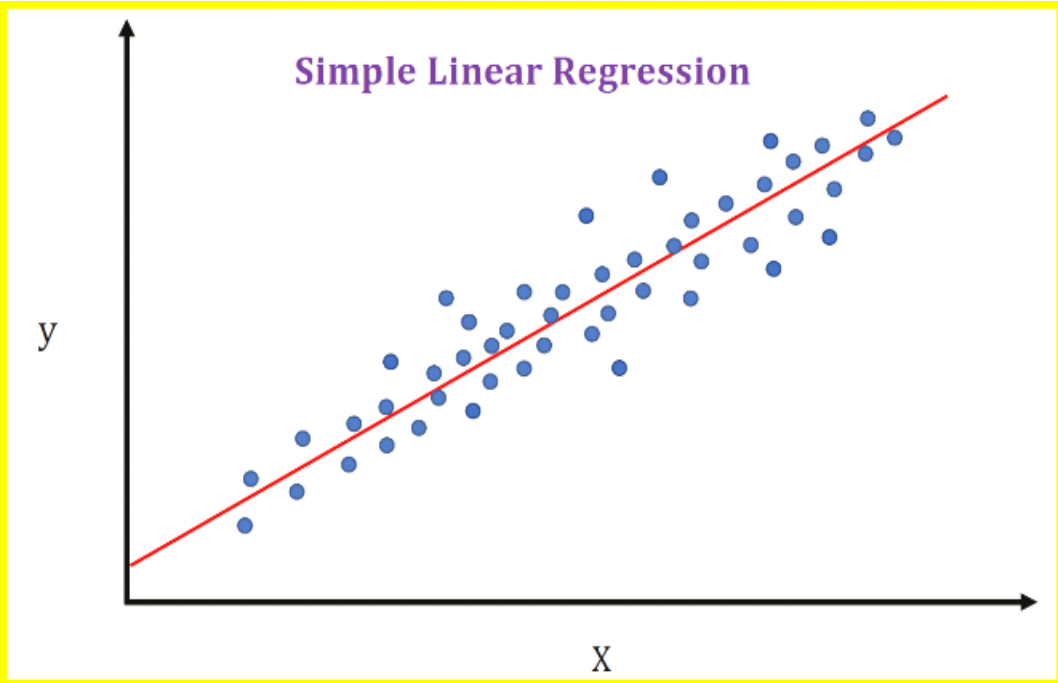
3. Linear regression

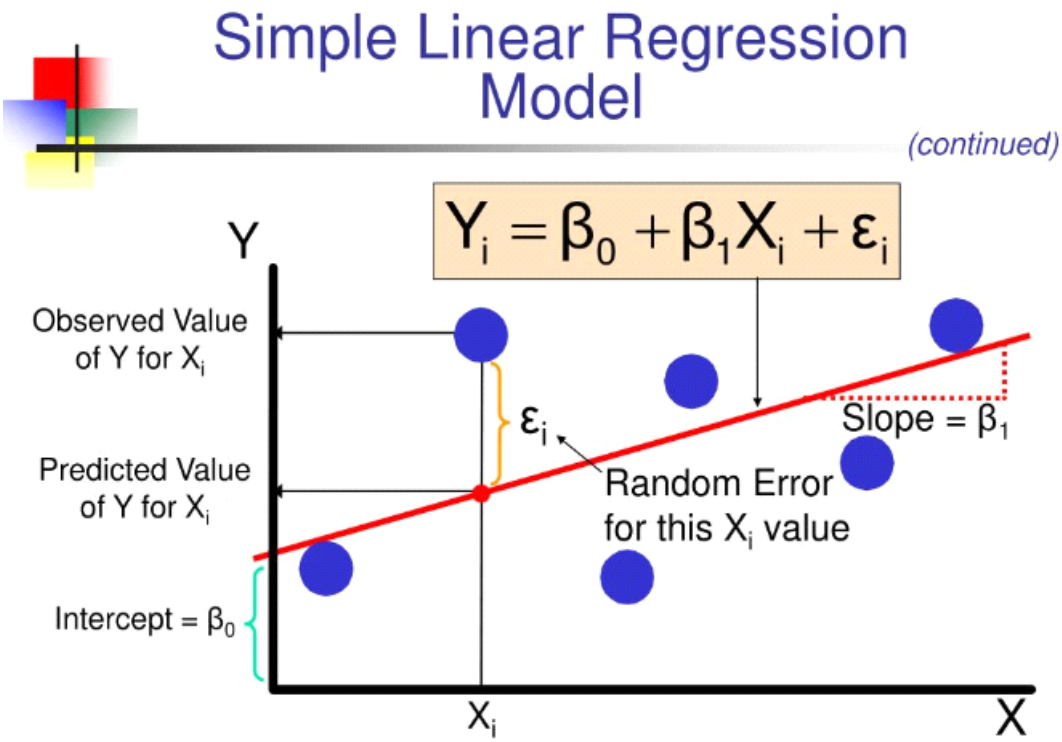
Linear regression is a supervised learning algorithm used for predicting and forecasting values that fall within a continuous range, such as sales numbers or housing prices. It is a technique derived from statistics and is commonly used to establish a relationship between an input variable (X) and an output variable (Y) that can be represented by a straight line.

In simple terms, linear regression takes a set of data points with known input and output values and finds the line that best fits those points. This line, known as the "regression line," serves as a predictive model. By using this line, we can estimate or predict the output value (Y) for a given input value (X).

Linear regression is primarily used for predictive modeling rather than categorization. It is useful when we want to understand how changes in the input variable affect the output variable. By analyzing the slope and intercept of the regression line, we can gain

insights into the relationship between the variables and make predictions based on this understanding.



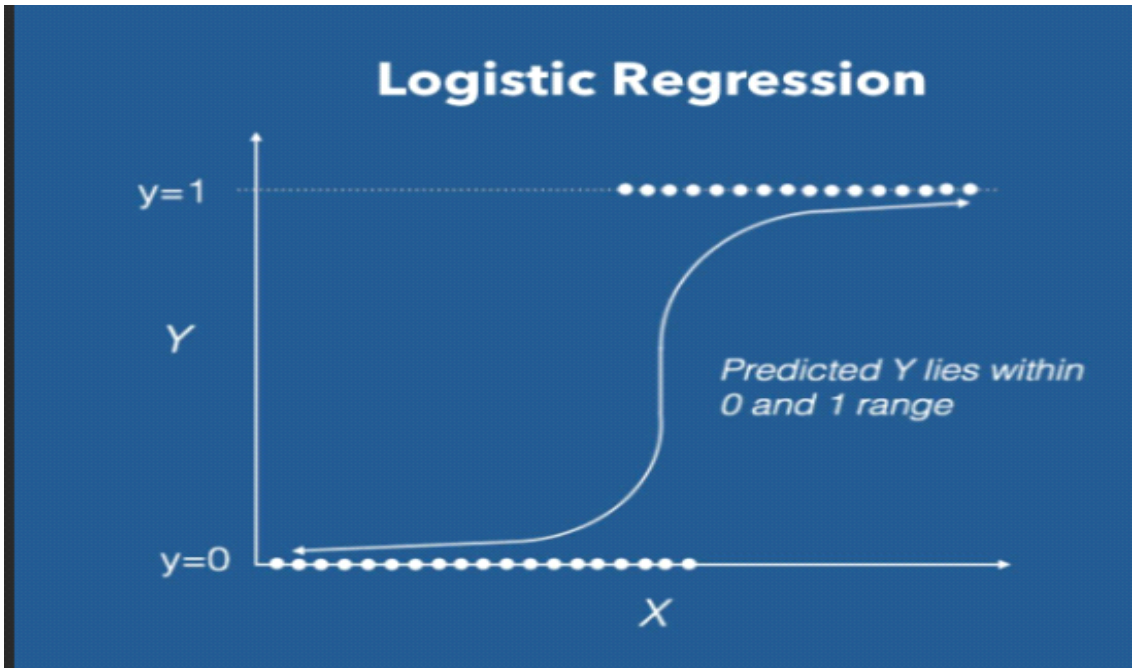


4. Logistic regression

Logistic regression, also known as "logit regression," is a supervised learning algorithm primarily used for binary classification tasks. It is commonly employed when we want to determine whether an input belongs to one class or another, such as deciding whether an image is a cat or not a cat.

Logistic regression predicts the probability that an input can be categorized into a single primary class. However, in practice, it is commonly used to group outputs into two categories: the primary class and not the primary class. To accomplish this, logistic regression creates a threshold or boundary for binary classification. For example, any output value between 0 and 0.49 might be classified as one group, while values between 0.50 and 1.00 would be classified as the other group.

Consequently, logistic regression is typically used for binary categorization rather than predictive modeling. It enables us to assign input data to one of two classes based on the probability estimate and a defined threshold. This makes logistic regression a powerful tool for tasks such as image recognition, spam email detection, or medical diagnosis where we need to categorize data into distinct classes.



5. Naive Bayes

Naive Bayes is a set of supervised learning algorithms used to create predictive models for binary or multi-classification tasks. It is based on Bayes' Theorem and operates on conditional probabilities, which estimate the likelihood of a classification based on the combined factors while assuming independence between them.

Let's consider a program that identifies plants using a Naive Bayes algorithm. The algorithm takes into account specific factors such as perceived size, color, and shape to categorize images of plants. Although each of these factors is considered independently, the algorithm combines them to assess the probability of an object being a particular plant.

Naive Bayes leverages the assumption of independence among the factors, which simplifies the calculations and allows the algorithm to work efficiently with large datasets. It is particularly well-suited for tasks like document classification, email spam filtering, sentiment analysis, and many other applications where the factors can be considered separately but still contribute to the overall classification.

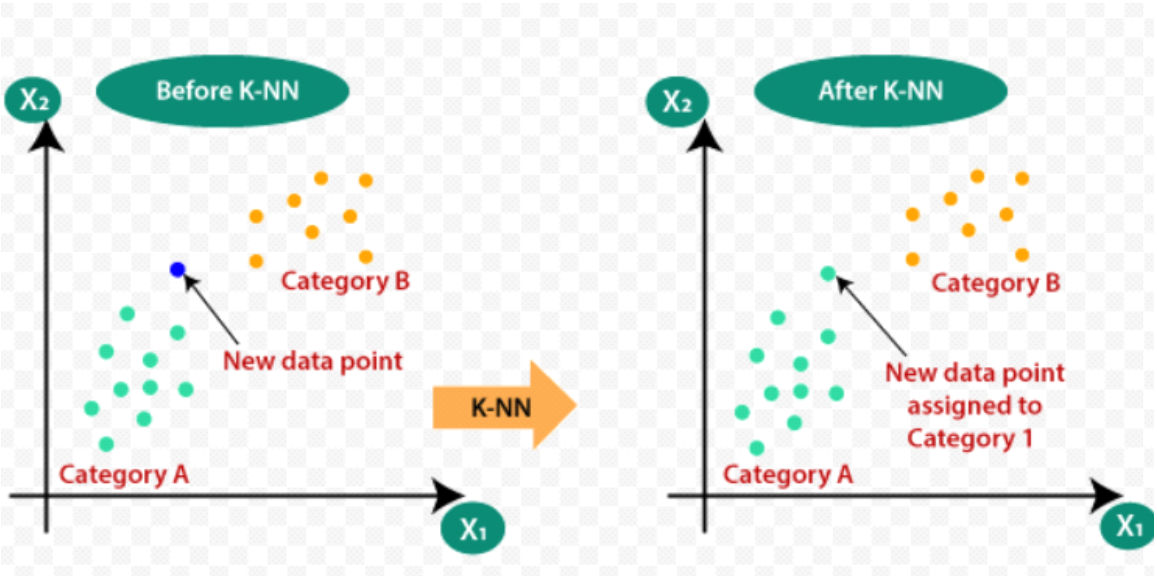
6. K-nearest neighbor (KNN)

K-nearest neighbor (KNN) is a supervised learning algorithm commonly used for classification and predictive modeling tasks. The name "K-nearest neighbor" reflects the algorithm's approach of classifying an output based on its proximity to other data points on a graph.

Let's say we have a dataset with labeled points, some marked as blue and others as green we want to classify a new data point, KNN looks at its nearest neighbors in the graph. The "K" in KNN refers to the number of nearest neighbors considered. For example, if K is set to 5, the algorithm looks at the 5 closest points to the new data point.

Based on the majority of the labels among the K nearest neighbors, the algorithm assigns a classification to the new data point. For instance, if most of the nearest neighbors are blue points, the algorithm classifies the new point as belonging to the blue group.

Additionally, KNN can also be used for prediction tasks. Instead of assigning a class label, KNN can estimate the value of an unknown data point based on the average or median of its K nearest neighbors.



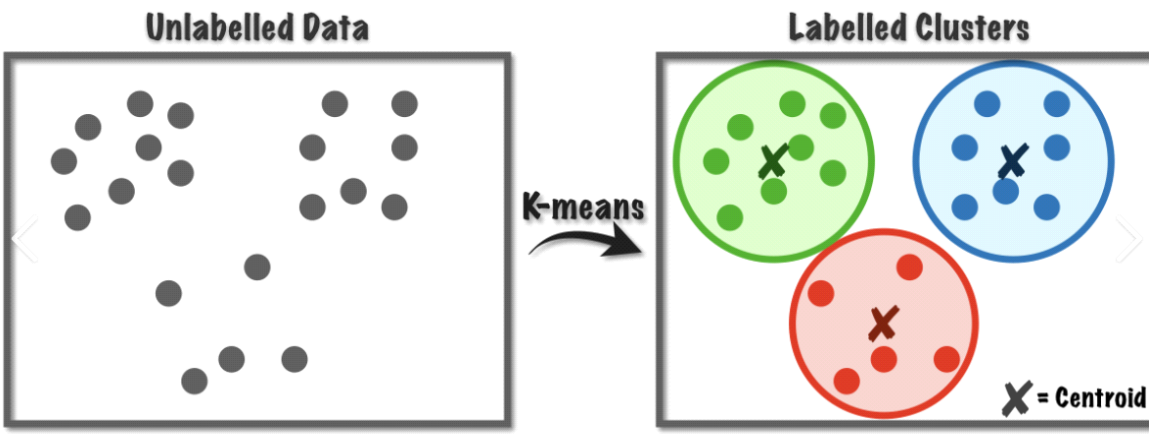
7. K-means

K-means is an unsupervised learning algorithm commonly used for clustering and pattern recognition tasks. It aims to group data points based on their proximity to one another. Similar to K-nearest

neighbor (KNN), K-means utilizes the concept of proximity to identify patterns or clusters in the data.

Each of the clusters is defined by a centroid, a real or imaginary center point for the cluster. K-means is useful on large data sets, especially for clustering, though it can falter when handling outliers.

K-means is particularly useful for large datasets and can provide insights into the inherent structure of the data by grouping similar points together. It has applications in various fields such as customer segmentation, image compression, and anomaly detection.

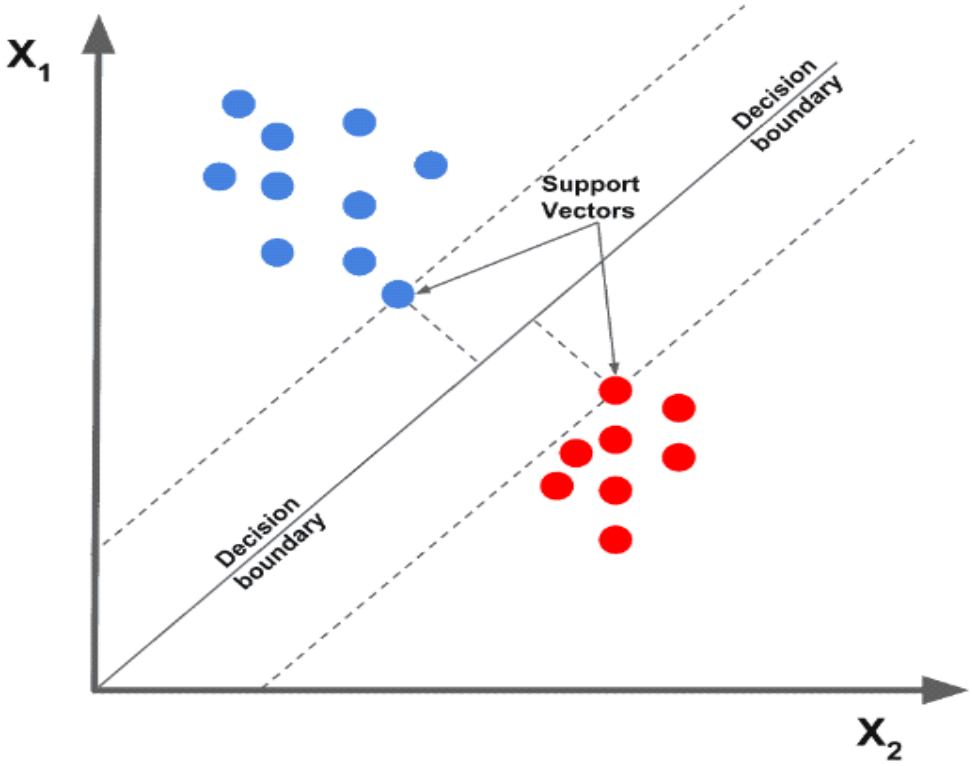


8. Support vector machine (SVM)

A support vector machine (SVM) is a supervised learning algorithm commonly used for classification and predictive modeling tasks. SVM algorithms are

popular because they are reliable and can work well even with a small amount of data. SVM algorithms work by creating a decision boundary called a "hyperplane." In two-dimensional space, this hyperplane is like a line that separates two sets of labeled data.

The goal of SVM is to find the best possible decision boundary by maximizing the margin between the two sets of labeled data. It looks for the widest gap or space between the classes. Any new data point that falls on either side of this decision boundary is classified based on the labels in the training dataset.



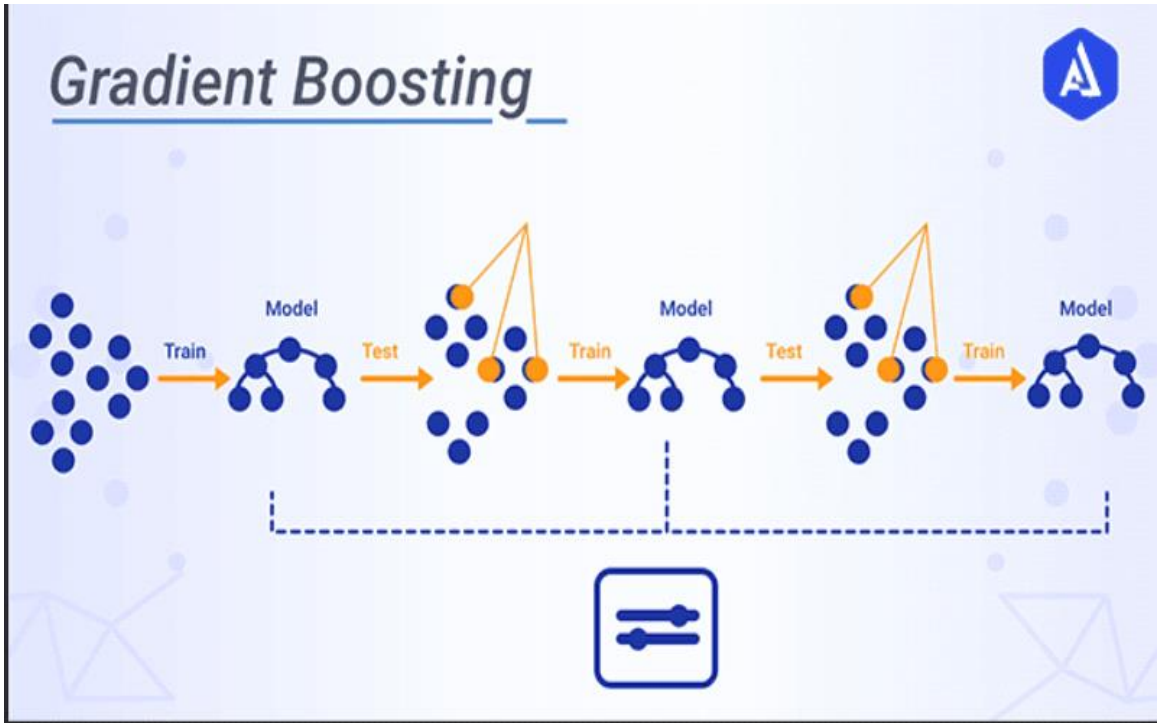
9. Gradient boosting

Gradient boosting algorithms employ an ensemble method, which means they create a series of "weak" models that are iteratively improved upon to form a strong predictive model. The iterative process gradually reduces the errors made by the models, leading to the generation of an optimal and accurate final model.

The algorithm starts with a simple, naive model that may make basic assumptions, such as classifying data based on whether it is above or below the mean. This initial model serves as a starting point.

In each iteration, the algorithm builds a new model that focuses on correcting the mistakes made by the previous models. It identifies the patterns or relationships that the previous models struggled to capture and incorporates them into the new model.

Gradient boosting is effective in handling complex problems and large datasets. It can capture intricate patterns and dependencies that may be missed by a single model. By combining the predictions from multiple models, gradient boosting produces a powerful predictive model



In conclusion

deep learning and machine learning, in general, have shown significant progress and potential in solving complex real-world problems. The algorithms discussed in this article are just a few examples of the many tools available in the field, and researchers continue to develop new and innovative techniques to improve performance and expand the scope of applica*tions*.

***In the next part we will talk about our machine learning model and our dataset specially: -**

Data collection

We searched for a dataset to use in our model and our project, and we found many datasets but most of the were small datasets and will be bad in learning, so we decided to use " Breast Cancer Wisconsin (Diagnostic) Data Set. "

Because it was big enough to be learnt by the machine learning models.

```
In [ ]:

In [4]: print (dataset.shape)

        (569, 33)

In [ ]:

In [5]: print (dataset.head(5)) # here we see that all features of data is int unless the diagnosis features

           id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
        0   842302         M        17.99         10.38          122.80     1001.0
        1   842517         M        20.57         17.77          132.90     1326.0
        2  84300903        M        19.69         21.25          130.00     1203.0
        3  84348301        M        11.42         20.38           77.58      386.1
        4  84358402        M        20.29         14.34          135.10     1297.0

           smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
        0          0.11840           0.27760          0.3001              0.14710
        1          0.08474           0.07864          0.0869              0.07017
        2          0.10960           0.15990          0.1974              0.12790
        3          0.14250           0.28390          0.2414              0.10520
        4          0.10030           0.13280          0.1980              0.10430

           ...  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
        0  ...          17.33           184.60      2019.0            0.1622
        1  ...          23.41           158.80      1956.0            0.1238
        2  ...          25.53           152.50      1709.0            0.1444
        3  ...          26.50            98.87       567.7            0.2098
        4  ...          16.67           152.20      1575.0            0.1374

           compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
        0             0.6656           0.7119                0.2654          0.4601
        1             0.1866           0.2416                0.1860          0.2750
        2             0.4245           0.4504                0.2430          0.3613
        3             0.8663           0.6869                0.2575          0.6638
        4             0.2050           0.4000                0.1625          0.2364

           fractal_dimension_worst  Unnamed: 32
        0                  0.11890          NaN
        1                  0.08902          NaN
        2                  0.08758          NaN
        3                  0.17300          NaN
        4                  0.07678          NaN

        [5 rows x 33 columns]

In [ ]:
```

***The dataset has 569 rows (attribute) and 33 columns (features).**

Data preprocessing

Data preprocessing is a step in the data mining and data analysis process that takes raw data and transforms it into a format that can be understood and analyzed by computers and machine learning.

Raw, real-world data in the form of text, images, video, etc., is messy. Not only may it contain errors and inconsistencies, but it is often incomplete, and doesn't have a regular, uniform design.

Machines like to process nice and tidy information – they read data as 1s and 0s. So, calculating structured data, like whole numbers and percentages is easy. However, unstructured data, in the form of text and images must first be cleaned and formatted before analysis.

So, to prevent these problems we so some data preprocessing techniques in our project to build a good model: -

- First we search for a missing values to handle it: -

```
In [40]:
# Assuming your data is stored in a DataFrame called 'dataset'

# Check for missing values in the entire DataFrame
missing_values = dataset.isnull().sum()
print(missing_values)

# Check for missing values in a specific column
missing_values_in_column = dataset['Column_Name'].isnull().sum()
print(missing_values_in_column)
```

```
id                         0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
diagnosis                  0
dtype: int64
```

- Here above we see that the dataset was clean and doesn't have any missing values so it's ready to use in the model.

- Text data requires special preparation before you can start using it for predictive modeling.

- So after that we prepare the data by encoding some features to be easy for the model to use, we have different encoders to use like (one_hot_encoder) and (label_encoder), here we used the 'label_encoder':-

```
In [7]: # Importing LabelEncoder from Sklearn
        # library from preprocessing Module.
        from sklearn.preprocessing import LabelEncoder

        # Creating a instance of label Encoder.
        le = LabelEncoder()

        # Using .fit_transform function to fit label
        # encoder and return encoded label
        label = le.fit_transform(dataset['diagnosis'])

        # printing label
        label
```

```
Out[7]: array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
               1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
               1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1,
               0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1,
               0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
               0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1,
               1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
               0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1,
               1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1,
               0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
               0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1,
               1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
               1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1,
               0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
               0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0,
               0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
               0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
               0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0,
               0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0])
```

- As we see above we use label encoder to convert all text data which was in the "diagnosis" feature (the label) to numeric data zeros and ones; one for "m" Malignant and zero for "b" Benign.

- After that we will remove some data which we don't need, and it has no effect on the label or model learning (there is no relation between this feature and the other features) example below: -

```
In [12]: dataset.iloc[:,[0]] # not used in training or testing
Out[12]:
```

	id
0	842302
1	842517
2	84300903
3	84348301
4	84358402
...	...
564	926424
565	926682
566	926954
567	927241
568	92751

569 rows × 1 columns

Features selection

Feature selection is also called variable selection or attribute selection.

It is the automatic selection of attributes in your data (such as columns in tabular data) that are most relevant to the predictive modeling problem you are working on.

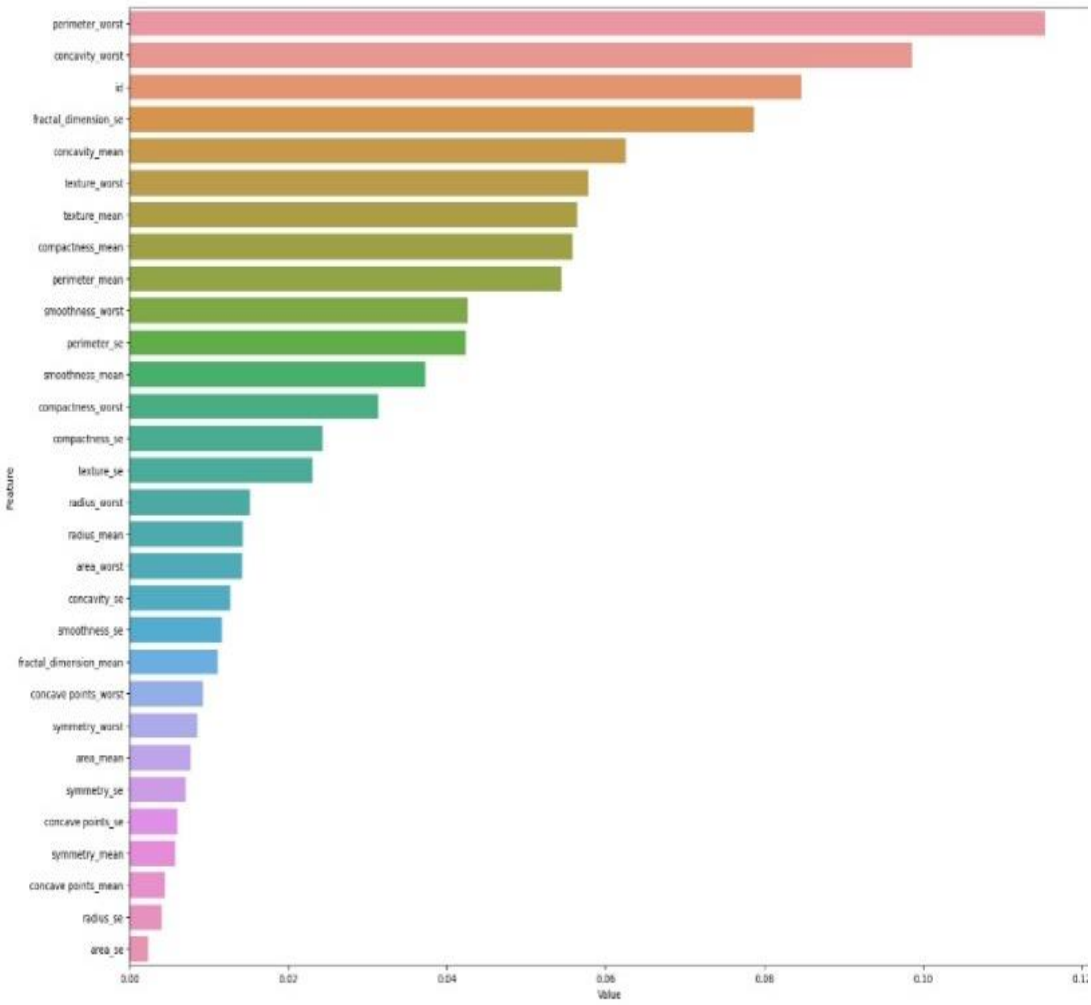
feature selection… is the process of selecting a subset of relevant features for use in model construction.

Feature selection is different from dimensionality reduction. Both methods seek to reduce the number of attributes in the dataset, but a dimensionality reduction method do so by creating new combinations of attributes, whereas feature selection methods include and exclude attributes present in the data without changing them.

Examples of dimensionality reduction methods include Principal Component Analysis, Singular Value Decomposition and Sammon's Mapping.

Finally: ***The objective of variable selection is three-fold: improving the prediction performance of the predictors, providing faster and more cost-effective predictors, and providing a better understanding of the underlying process that generated the data.***

```
In [42]: feature_imp = pd.DataFrame(sorted(zip(rf_model.feature_importances_,X.columns)), columns=['Value','Feature'])

         plt.figure(figsize=(20, 15))
         sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value", ascending=False))

Out[42]: <Axes: xlabel='Value', ylabel='Feature'>
```



- Here we see the features importance graph,

- In our project we decided to use only 10 features from the all features of the dataset however the data importance, the features we decided to use are: -

1- The "radius_mean" is one of the features in breast cancer datasets that represents the mean of distances from the center to points on the perimeter of the tumor.

2- The "texture_mean" is another feature in breast cancer datasets that represents the mean value of the gray-scale intensities of the pixels in the image of the tumor.

3- The "perimeter_mean" is another feature commonly found in breast cancer datasets. It represents the mean value of the perimeter (the distance around the tumor) of the breast mass.

4- The "area_mean" is another common feature in breast cancer datasets. It represents the mean value of the area (the total number of pixels) of the tumor mass.

5- The "smoothness_mean" is another feature commonly found in breast cancer datasets. It represents the mean variation in the local smoothness of the tumor boundaries.

6- The "compactness_mean" is another feature commonly present in breast cancer datasets. It

represents the mean value of the compactness of the tumor, which is calculated as the perimeter squared divided by the area.

7- The "concavity_mean" is another feature commonly found in breast cancer datasets. It represents the mean severity of concave portions of the contour of the tumor.

8- The "concave points_mean" is another feature commonly present in breast cancer datasets. It represents the mean number of concave portions of the contour that are formed by the tumor.

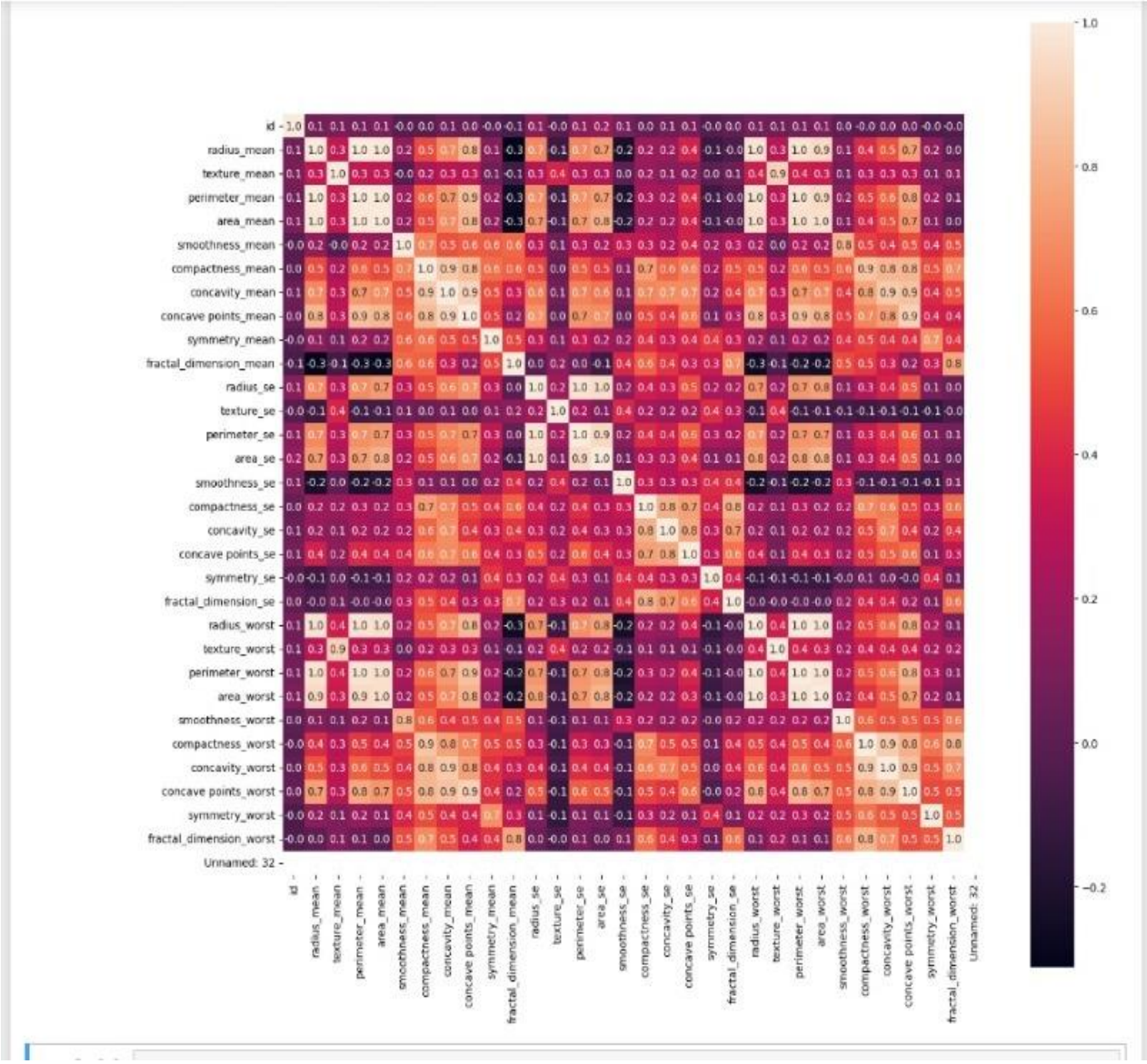
9- The "symmetry_mean" is another feature commonly found in breast cancer datasets. It represents the mean value of the symmetry of the tumor, which measures how symmetric the contour of the tumor is.

10- The "fractal_dimension_mean" is another feature commonly present in breast cancer datasets. It represents the mean value of the fractal dimension of the tumor, which characterizes the complexity of the tumor boundary.

- This are all features we will build our project on and we will learn them to the machine learning model and this are the features which we will ask the patient or the doctor to enter to the application in the final stages to classify the tumor type.

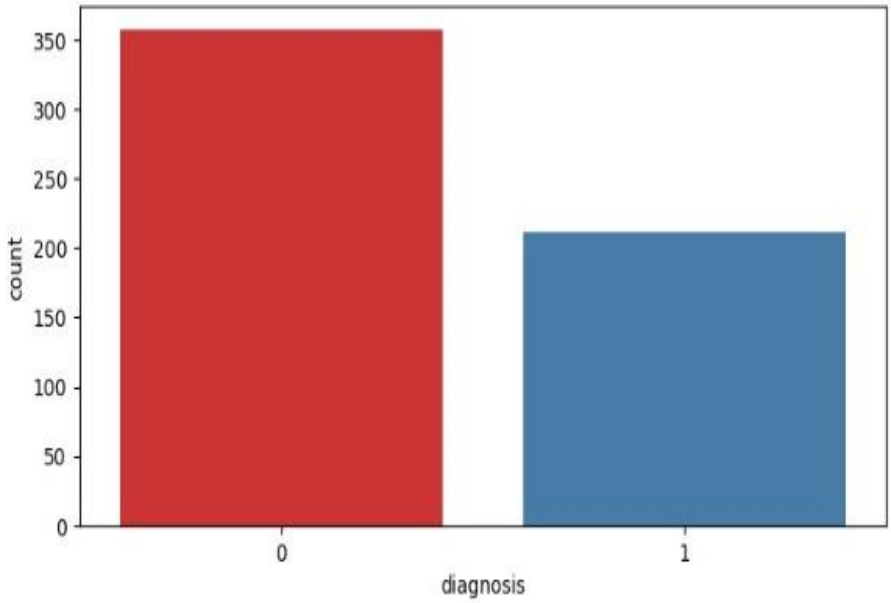
Data visualization

Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data. Additionally, it provides an excellent way for employees or business owners to present data to non-technical audiences without confusion.



- This was a heat_map for the whole data (before feature selection) above.

```
In [28]: plt.figure(figsize=(8, 4))
         sns.countplot(x=dataset['diagnosis'], palette='Set1')

         # Count number of observations in each class
         benign, malignant = dataset['diagnosis'].value_counts()
         print('No. Benign: ', benign)
         print('No. Malignant : ', malignant)
         print('')
         print('% of cells labeled Benign', round(benign / len(dataset) * 100, 2), '%')
         print('% of cells labeled Malignant', round(malignant / len(dataset) * 100, 2), '%')

         No. Benign:  357
         No. Malignant :  212

         % of cells labeled Benign 62.74 %
         % of cells labeled Malignant 37.26 %
```

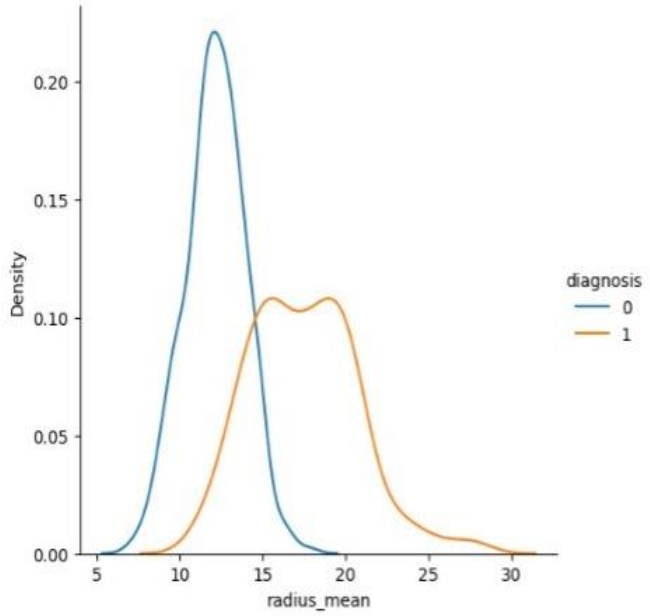


- here we can see the amount of malignant and also the amount of benign and the percentage of them.

- Now we will do some graphs for the selected features in order: -

1- radius_mean (very effective):

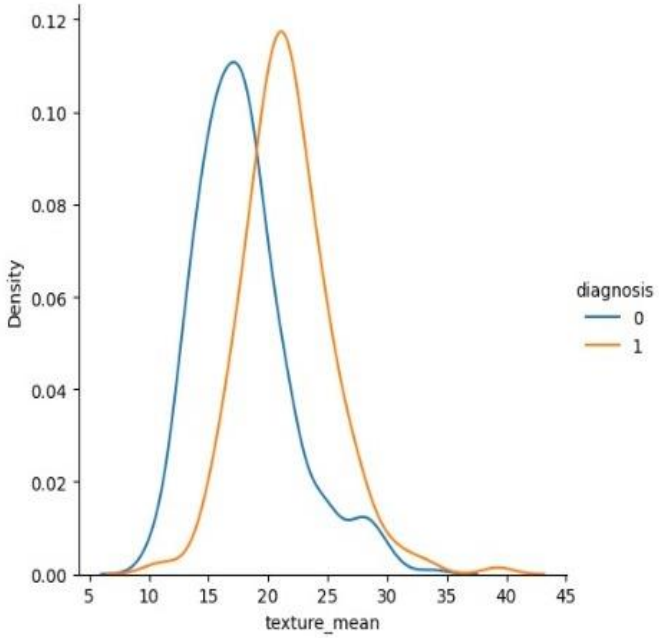
```
In [ ]:

In [28]: sns.FacetGrid(dataset,hue='diagnosis',height=5).map(sns.kdeplot,"radius_mean").add_legend()
         plt.show()
```




```
In [ ]:
```

2- texture_mean:

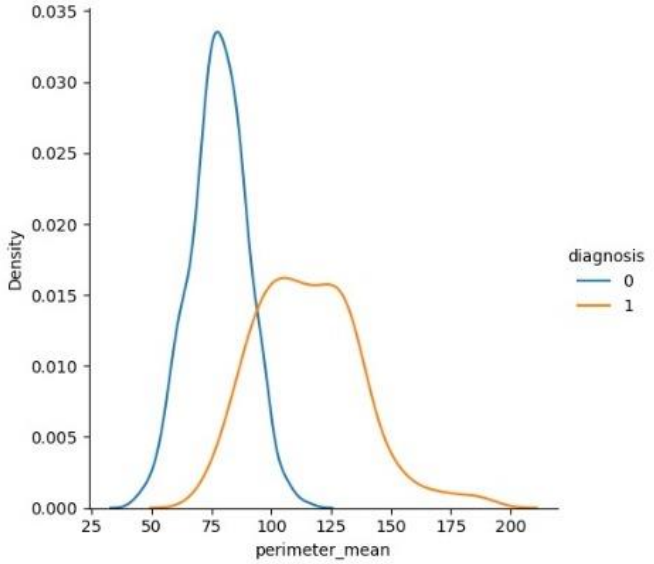
```
In [29]: sns.FacetGrid(dataset,hue='diagnosis',height=5).map(sns.kdeplot,"texture_mean").add_legend()
         plt.show()
```




```
In [ ]:
```

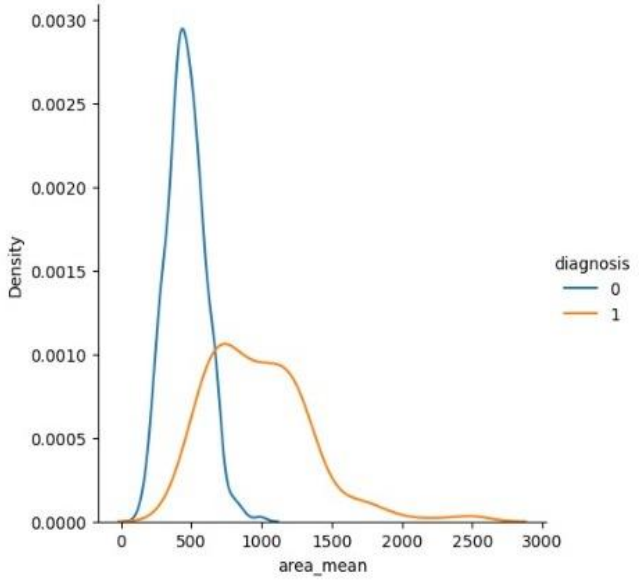
3- perimeter_mean (very effective):

```
In [ ]:

In [30]: sns.FacetGrid(dataset,hue='diagnosis',height=5).map(sns.kdeplot,"perimeter_mean").add_legend()
         plt.show()
```



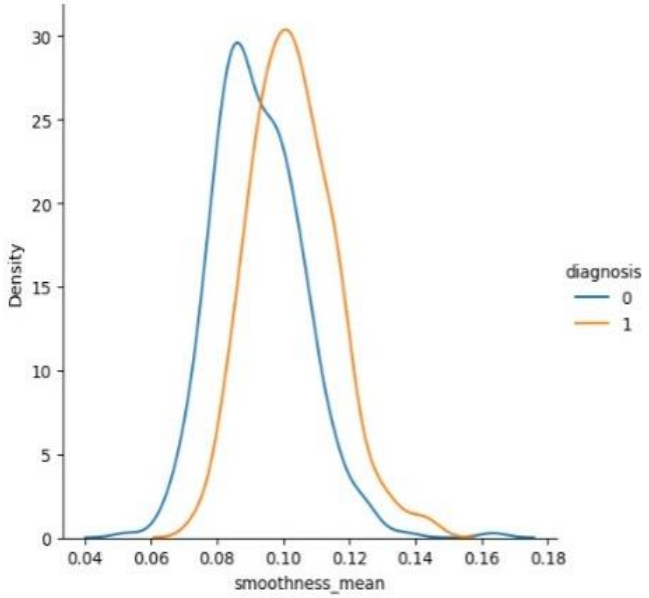
4- area_mean (very effective):

```
In [ ]:

In [31]: sns.FacetGrid(dataset,hue='diagnosis',height=5).map(sns.kdeplot,"area_mean").add_legend()
         plt.show()
```



5- smoothness_mean:

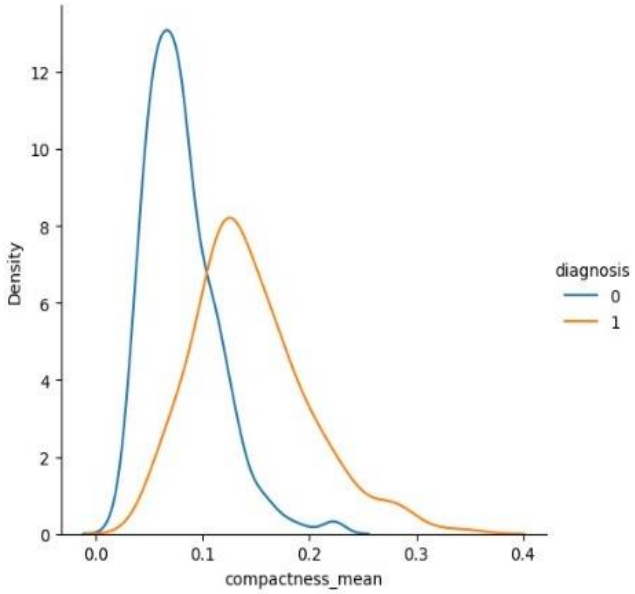
```
In [ ]:

In [32]: sns.FacetGrid(dataset,hue='diagnosis',height=5).map(sns.kdeplot,"smoothness_mean").add_legend()
         plt.show()
```




```
In [ ]:
```

6- compactness_mean (very effective):

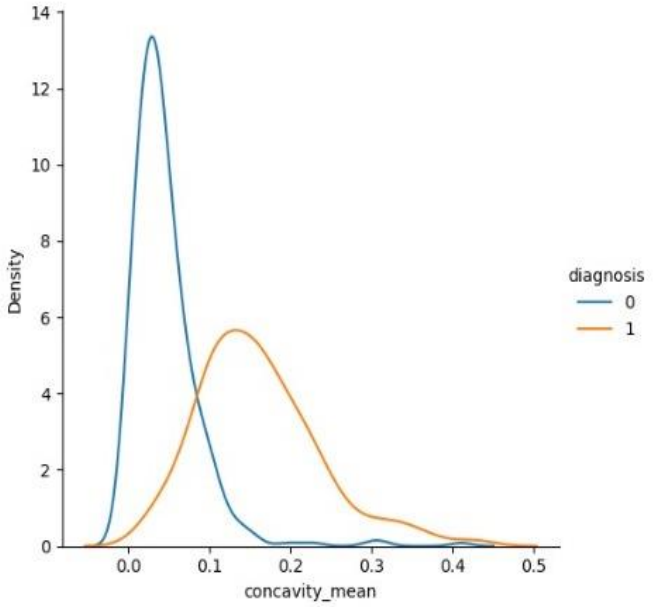
```
In [33]: sns.FacetGrid(dataset,hue='diagnosis',height=5).map(sns.kdeplot,"compactness_mean").add_legend()
         plt.show()
```




```
In [ ]:
```

7- concavity_mean (very effective):

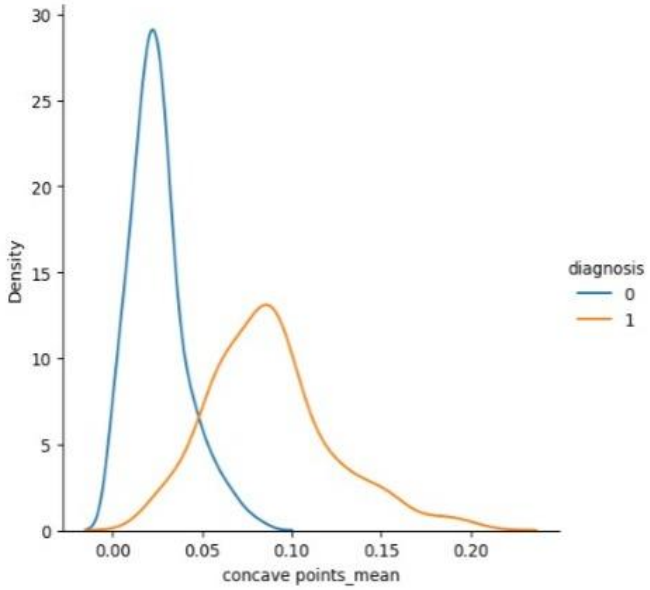
```
In [ ]:

In [35]: sns.FacetGrid(dataset,hue='diagnosis',height=5).map(sns.kdeplot,"concavity_mean").add_legend()
         plt.show()
```




```
In [ ]:
```

8- concave points_mean (very effective):

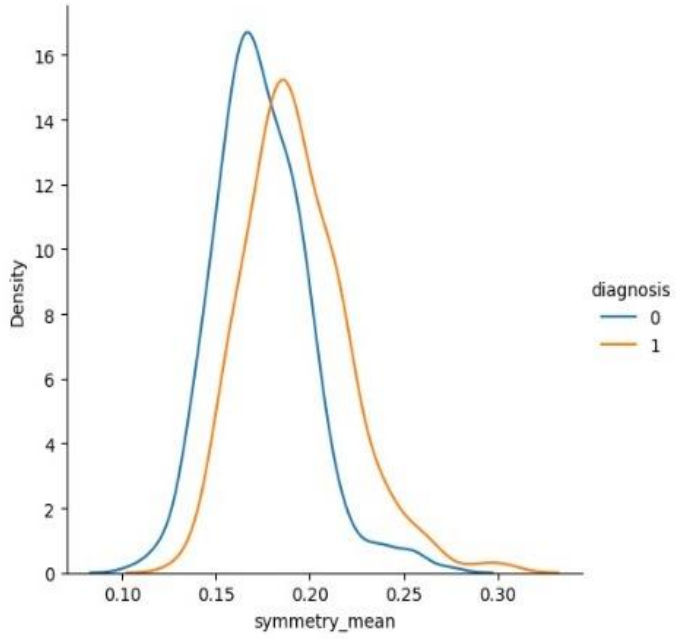
```
In [ ]:

In [36]: sns.FacetGrid(dataset,hue='diagnosis',height=5).map(sns.kdeplot,"concave points_mean").add_legend()
         plt.show()
```




```
In [ ]:
```

9- symmetry_mean:

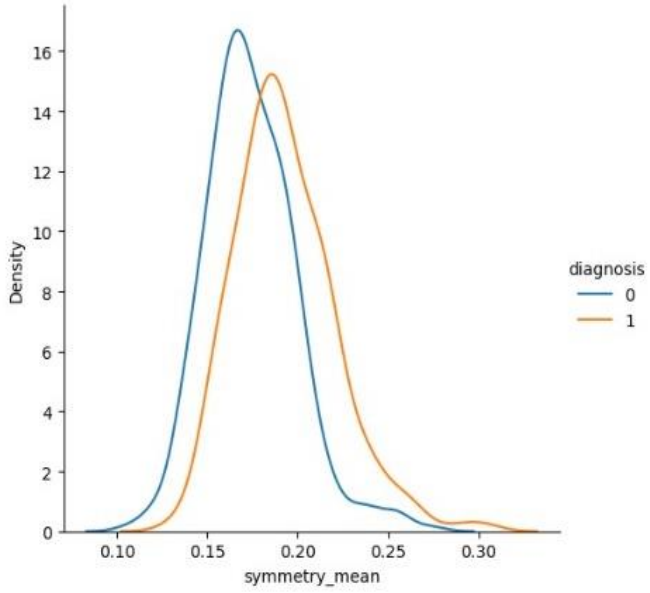
```
In [ ]:

In [37]: sns.FacetGrid(dataset,hue='diagnosis',height=5).map(sns.kdeplot,"symmetry_mean").add_legend()
         plt.show()
```




```
In [ ]:
```

10- fractal_dimension_mean:

```
In [ ]:

In [37]: sns.FacetGrid(dataset,hue='diagnosis',height=5).map(sns.kdeplot,"fractal_dimension_mean").add_legend()
         plt.show()
```




```
In [ ]:
```

Machine learning model and the classifier

Machine learning classifiers are algorithms used to classify or predict the target variable based on input features.

We searched for many algorithms to use in our project, and we tried many classifiers many times to select the best one to use it in the model.

Types of classifiers: -

1- Decision Trees: A non-parametric classifier that makes decisions by recursively partitioning the input space based on feature values. Decision trees can handle both classification and regression tasks.

2- Random Forest: An ensemble classifier that combines multiple decision trees. It reduces overfitting and improves generalization by averaging the predictions of individual trees.

3- Logistic Regression: A linear classifier used for binary classification problems. It models the probability of the input belonging to a particular class using a logistic function.

4- Support Vector Machines (SVM): A classifier that finds an optimal hyperplane in a high-dimensional space to separate different classes. SVMs can handle both linear and non-linear classification problems.

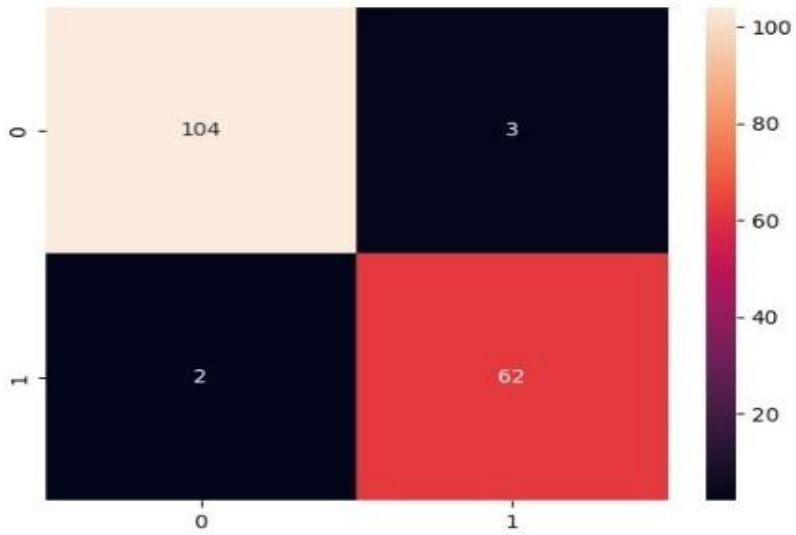
5- K-Nearest Neighbors (KNN): A lazy learning classifier that assigns labels to new instances based on the majority vote of their k nearest neighbors in the training set.

- And there are some other types of classifiers like: Gradient Boosting Machines (GBM), Deep Learning Classifiers , Neural Networks , Ensemble Methods: These methods combine multiple classifiers to improve performance .

*** In our project we decided to use the random forest algorithm (classifier) because it was suitable for the data set and achieve a good performance with a good accuracy and good with some other measures .**

```
In [23]: #Import scikit-learn metrics module for accuracy calculation
         from sklearn import metrics

         from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report, confusion_matrix

         f1 = f1_score(y_test, y_pred)

         prec = precision_score(y_test, y_pred)

         recall = recall_score(y_test, y_pred)

         Accuracy = metrics.accuracy_score(y_pred , y_test)

         # Model Accuracy: how often is the classifier correct?
         print('Accuracy  :',Accuracy)
         print ('precision :',prec )
         print ('recall    :',recall )
         print ('f1_score  :',f1 )

         end_time = datetime.now()
         print('Duration  : {}'.format(end_time - start_time))

         Accuracy  : 0.9824561403508771
         precision : 0.9830508474576272
         recall    : 0.9666666666666667
         f1_score  : 0.9747899159663865
         Duration  : 0:00:00.663512
```

```
In [21]: #confusion matrix

from sklearn.metrics import confusion_matrix

conf = confusion_matrix(y_pred,y_test)

print('confusion matrix',conf)

cm_2 = conf
sns.heatmap(cm_2,annot=True,fmt="d")

confusion matrix [[104   3]
 [  2  62]]

Out[21]: <Axes: >
```



- Here we can see the measures and the confusion matrix to know the performance of the classifier (algorithm) which was "random_forest".

- we also did a comparison between some classifier:

Algorithm	Accuracy	precision	recall	F1_Score	duration
knn	0.909	0.925	0.917	0.941	0:00:00.2 8315
svm	0.9705	0.96345	0.9588	0.96015	0:00:00.6 365
Random f	0.9726	0.9737	0.9532	0.9612	0:00:00.6 3775
logistic r	0.9426	0.95295	0.8958	0.92215	0:00:00.9 1825
decision t	0.93455	0.92795	0.89755	0.17735	0:00:00.6 794

- in this table we calculate the mean (average) of Accuracy, precision, recall ,F1_Score ,duration for 5 algorithms.

Saving the model

-After we set the classifier and training it and evaluation we need to save the model ; so we used the "joblib" library to doing that.

Joblib is a Python library that provides tools for serialization (saving and loading) of Python objects, particularly for objects that have large data structures such as machine learning models.

Joblib is commonly used for efficient persistence and retrieval of machine learning models, especially those trained on large datasets. It offers functionality to store Python objects, such as scikit-learn models, efficiently on disk, and retrieve them later for reuse. It is often used in combination with popular machine learning libraries like scikit-learn.

***This is our code for saving the model: -**

```
In [30]: import joblib

In [31]: joblib.dump(rf_model,"random forest algorithm final")
Out[31]: ['random forest algorithm final']

In [ ]:
```

***And this is also the code for loading the model: -**

```
In [32]: joblib.load("random forest algorithm final")
Out[32]: RandomForestClassifier
         RandomForestClassifier(max_depth=7, max_features=2, min_samples_split=25,
                                random_state=42)

In [ ]:
```

Creating the API, uploading the model.

API stands for Application Programming Interface. It is a set of rules and protocols that allows different software applications to communicate and interact with each other. APIs define how different components of software systems should interact, specifying the methods, data formats, and rules for accessing and exchanging information.

APIs can be used to access functionality and data from external services or libraries, enabling developers to incorporate pre-built capabilities into their own applications. They provide a way for developers to interact with software components without needing to know the internal implementation details.

There are different types of APIs, including:

1- Web APIs: These are APIs that are exposed over the web and allow applications to communicate and exchange data over HTTP. Web APIs are commonly used for integration with web services, accessing

data from external sources, or building web-based applications.

2- Library APIs: These are APIs provided by libraries or frameworks that allow developers to use the functionality and features provided by the library. Library APIs define the interfaces, classes, and methods that developers can use to interact with the library.

3- Operating System APIs: These are APIs provided by operating systems that enable developers to interact with the underlying system resources and services. Operating system APIs allow applications to access hardware devices, manage files and processes, and perform system-level operations.

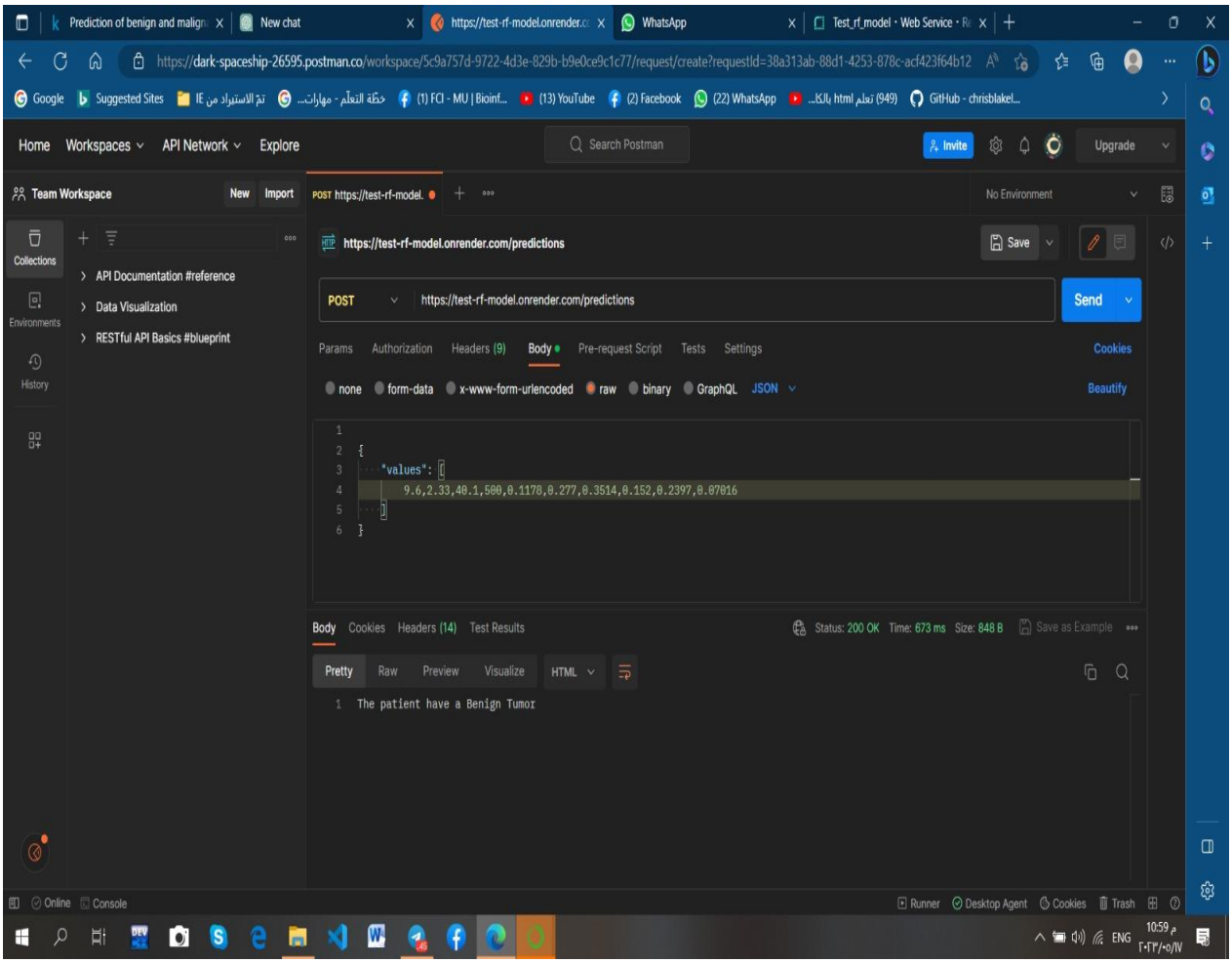
4- Database APIs: These are APIs provided by database management systems that allow developers to interact with databases. Database APIs define the methods and protocols for querying, inserting, updating, and deleting data in the database.

APIs are commonly used in software development to enable interoperability, code reusability, and integration with external services. They provide a standardized way for different software components to communicate and exchange data, making it easier

to build complex applications by leveraging existing functionalities and services.

***We used <u>Flask</u> library in python to create our web API and upload the model on a server.**

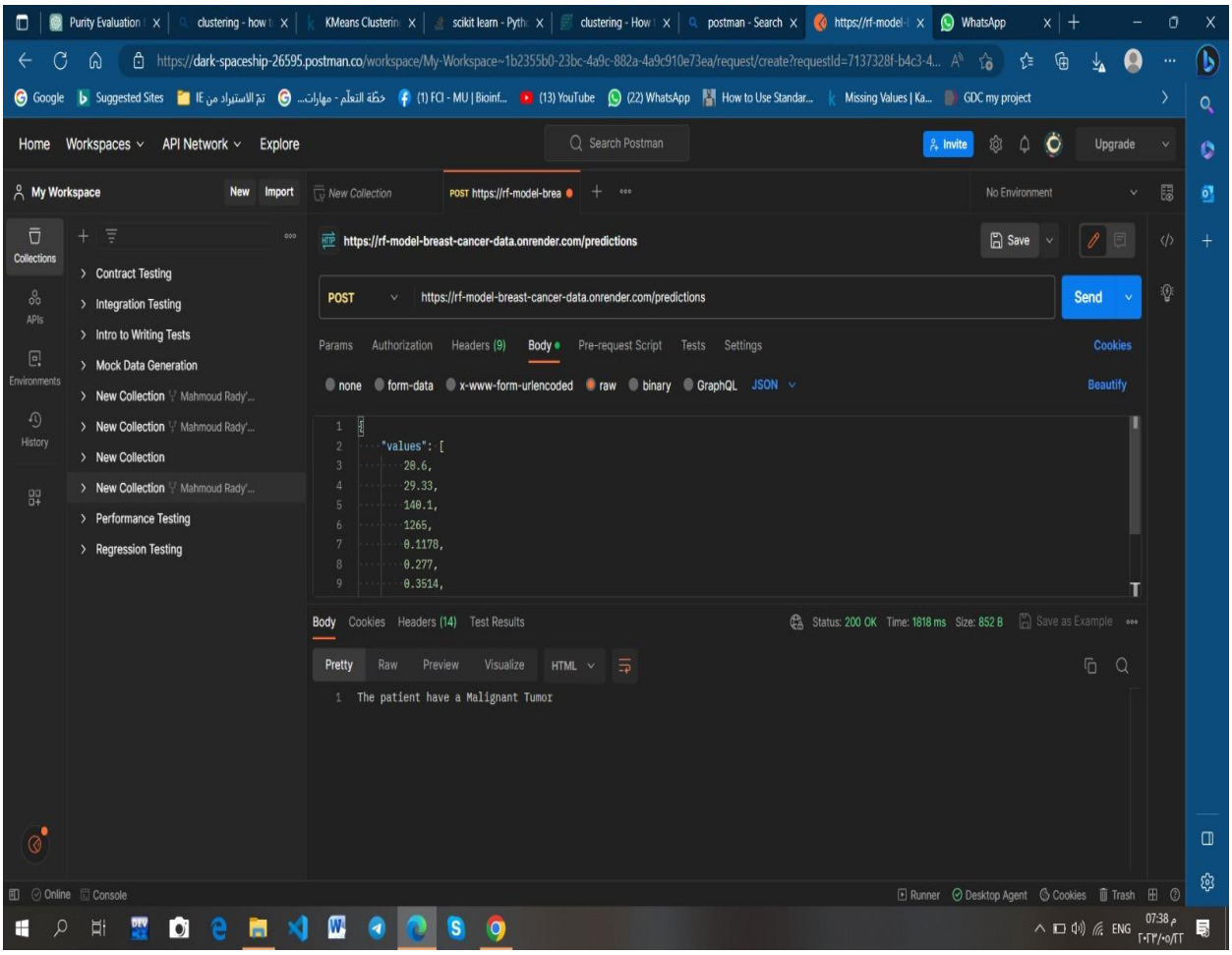
-After we create the Api and upload the model on a server; we test the API and the model using Postman website and Postman agent and we send some requests to the server and we can see the response in the figures below: -



*** Here we send a data as numbers in a variable called values from our dataset with label b and the response was that:**

"The patient has a Benign Tumor", and this is true.

***So, the API is working, and the model is good also.**



- here we can see other values with another label from the dataset and the result was true and the model is working well because the response was:
"The patient have a Malignant Tumor".

Flutter

Get Started

What is Flutter?
Flutter is Google's portable UI toolkit for crafting beautiful, natively compiled applications for mobile, web, and desktop from a single codebase. Flutter works with existing code, is used by developers and organizations around the world, and is free and open source.

-Test Drive

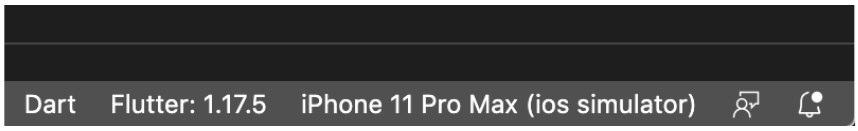
Create the app.

1. **Invoke View > Command Palette.**
2. **Type "flutter" and select Flutter: New Project.**
3. **Select Application.**
4. **Create or select the parent directory for the new project.**
5. **Enter a project name, such as my_app, and press Enter.**
6. **.. Wait for the project creation to complete and the main. Dart file to appear.**

The above commands create a Flutter project directory called my_app that contains a simple demo app that uses Material Components.

Run the app.

1-Locate the VS Code status bar (the blue bar at the bottom of the window):



2-Select a device from the Device Selector area. For details, see Quickly switching between Flutter devices.

- If no device is available, and you want to use a device simulator, click No Devices and click Start iOS Simulator to launch a simulator.

```
$ open -a simulator
```

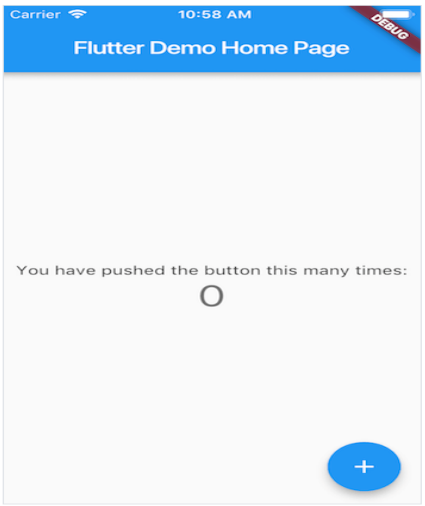
  - To setup a real device, follow the device-specific instructions on the Install page for your OS.

3.Invoke Run > Start Debugging or press F5.

4.Wait for the app to launch—progress is printed in the

Debug Console view.

After the app build is completed, you'll see the starter app on your device.



Try hot reload.
Flutter offers a fast development cycle with *Stateful Hot Reload*, the ability to reload the code of a live running app without

restarting or losing app state. Make a change to app source, tell your IDE or command-line tool that you want to hot reload, and see the change in your simulator, emulator, or device.

1. **Open lib/main. Dart.**
2. **Change the string.**

```
'You have pushed the button this many times'
```

to

```
'You have clicked the button this many times'
```

3. **Save your changes: invoke Save All, or click Hot Reload**

-from another platform

Flutter for Android developers

Views

What is the equivalent of a View in Flutter?

How is react-style, or *declarative*, programming different than the traditional imperative style? For a comparison, see Introduction to declarative UI.

In Android, the View is the foundation of everything that shows up on the screen. Buttons, toolbars, and inputs, everything is a View. In Flutter, the rough equivalent to a View is a Widget. Widgets don't map exactly to Android views, but while you're getting acquainted with how Flutter works you can think of them as "the way you declare and construct UI".

However, these have a few differences to a View. To start, widgets have a different lifespan: they are immutable and only exist until they need to be changed. Whenever widgets or their state change, Flutter's framework creates a new tree of widget instances. In comparison, an Android view is drawn once and does not redraw until invalidate is called.

Flutter's widgets are lightweight, in part due to their immutability. Because they aren't views themselves, and aren't directly drawing anything, but rather are a description of the UI and its semantics that get "inflated" into actual view objects under the hood.

Flutter includes the Material Components library. These are widgets that implement the Material Design guidelines. Material Design is a flexible design system optimized for all platforms, including iOS.

But Flutter is flexible and expressive enough to implement any design language. For example, on iOS, you can use the Cupertino widgets to produce an interface that looks like Apple's iOS design language.

For example:

How do I update widgets?
In Android, you update your views by directly mutating them. However, in Flutter, Widgets are immutable and are not updated directly, instead you must work with the widget's state.

This is where the concept of Stateful and Stateless widgets comes from. A Stateless Widget is just what it sounds like—a widget with no state information.

Stateless Widgets are useful when the part of the user interface you are describing does not depend on anything other than the configuration information in the object.

For example, in Android, this is similar to placing an Image View with your logo. The logo is not going to change during runtime, so use a Stateless Widget in Flutter.

If you want to dynamically change the UI based on data received after making an HTTP call or user interaction, then you must work with Stateful Widget and tell the Flutter framework that the widget's State has been updated so it can update that widget.

The important thing to note here is at the core both stateless and stateful widgets behave the same. They rebuild every frame; the difference is the Stateful Widget has a state object that stores state data across frames and restores it.

If you are in doubt, then always remember this rule: if a widget changes (because of user interactions, for example) it's stateful. However, if a widget reacts to change, the containing parent widget can still be stateless if it doesn't itself react to change.

The following example shows how to use a Stateless Widget. A common Stateless Widget is the Text widget. If you look at the implementation of the Text widget, you'll find that it subclasses Stateless Widget.

```
Text(
  'I like Flutter!',
  style: TextStyle(fontWeight: FontWeight.bold),
);
```

As you can see, the Text Widget has no state information associated with it, it renders what is passed in its constructors and nothing more. But what if you want to make "I Like Flutter" change dynamically, for example when clicking a FloatingActionButton?

To achieve this, wrap the Text widget in a Stateful Widget and update it when the user clicks the button.

```
import 'package:flutter/material.dart';

void main() {
  runApp(const SampleApp());
}

class SampleApp extends StatelessWidget {
  const SampleApp({super.key});

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Sample App',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: const SampleAppPage(),
    );
  }
}

class SampleAppPage extends StatefulWidget {
  const SampleAppPage({super.key});

  @override
  State<SampleAppPage> createState() => _SampleAppPageState();
}
```

```
class _SampleAppPageState extends State<SampleAppPage> {
  // Default value for toggle.
  bool toggle = true;
  void _toggle() {
    setState(() {
      toggle = !toggle;
    });
  }

  Widget _getToggleChild() {
    if (toggle) {
      return const Text('Toggle One');
    } else {
      return ElevatedButton(
        onPressed: () {},
        child: const Text('Toggle Two'),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Sample App'),
      ),
      body: Center(
        child: _getToggleChild(),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _toggle,
        tooltip: 'Update Text',
        child: const Icon(Icons.update),
    ),
  );
 }
}
```

-Dart language overview

Dart is a client-optimized language for developing fast apps on any platform. Its goal is to offer the most productive programming language for multi-platform development, paired with a <u>flexible execution runtime platform</u> for app frameworks.

Languages are defined by their *technical envelope*—the choices made during development that shape the capabilities and strengths of a language. Dart is designed for a technical envelope that is particularly suited to client development, prioritizing both development (sub-second stateful hot reload) and high-quality production experiences across a wide

variety of compilation targets (web, mobile, and desktop).

Dart also forms the foundation of Flutter. Dart provides the language and runtimes that power Flutter apps, but Dart also supports many core developer tasks like formatting, analyzing, and testing code.

Dart: The language

The Dart language is type safe; it uses static type checking to ensure that a variable's value *always* matches the variable's static type. Sometimes, this is referred to as sound typing. Although types are mandatory, type annotations are optional because of type inference. The Dart typing system is also flexible, allowing the use of a dynamic type combined with runtime checks, which can be useful during experimentation or for code that needs to be especially dynamic.

Dart has built-in sound null safety. This means values can't be null unless you say they can be. With sound null safety, Dart can protect you from null exceptions at runtime through static code analysis. Unlike many other null-safe languages, when Dart determines that a variable is non-nullable, that variable can never be null. If you inspect your running code in the debugger, you see that non-nullability is retained at runtime; hence *sound* null safety.

The following code sample showcases several Dart language features, including libraries, async calls, nullable and non-nullable types, arrow syntax, generators, streams, and getters. To learn more about the language, take the Dart language tour.

Dart Install SDK

```dart
import 'dart:math' show Random;

void main() async {
  print('Compute π using the Monte Carlo method.');
  await for (final estimate in computePi().take(100)) {
    print('π ≅ $estimate');
  }
}

/// Generates a stream of increasingly accurate estimates of π.
Stream<double> computePi({int batch = 100000}) async* {
  var total = 0; // Inferred to be of type int
  var count = 0;
  while (true) {
    final points = generateRandom().take(batch);
    final inside = points.where((p) => p.isInsideUnitCircle);

    total += batch;
    count += inside.length;
    final ratio = count / total;

    // Area of a circle is A = π·r², therefore π = A/r².
    // So, when given random points with x ∈ <0,1>,
    // y ∈ <0,1>, the ratio of those inside a unit circle
    // should approach π / 4. Therefore, the value of π
    // should be:
    yield ratio * 4;
  }
}

Iterable<Point> generateRandom([int? seed]) sync* {
  final random = Random(seed);
  while (true) {
    yield Point(random.nextDouble(), random.nextDouble());
  }
}

class Point {
  final double x;
  final double y;

  const Point(this.x, this.y);

  bool get isInsideUnitCircle => x * x + y * y <= 1;
}
```

Dart: The libraries

Dart has a rich set of core libraries, providing essentials for many everyday programming tasks:

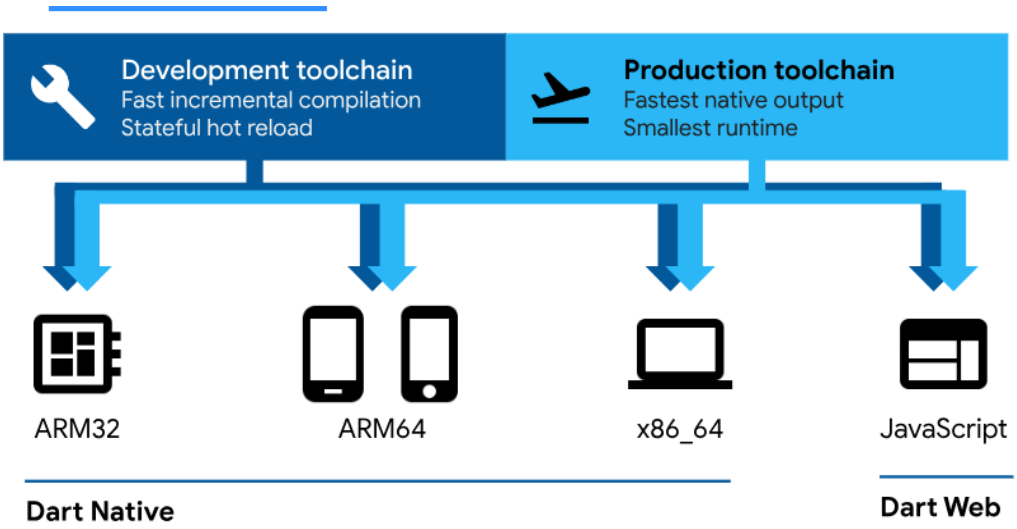
- **Built-in types, collections, and other core functionality for every Dart program (dart: core)**

- **Richer collection types such as queues, linked lists, hash maps, and binary trees (dart: collection)**
- **Encoders and decoders for converting between different data representations, including JSON and UTF-8 (dart: convert)**
- **Mathematical constants and functions, and random number generation (Dart: math)**
- **File, socket, HTTP, and other I/O support for non-web applications (dart: io)**
- **Support for asynchronous programming, with classes such as Future and Stream (dart:async)**
- **Lists that efficiently handle fixed-sized data (for example, unsigned 8-byte integers) and SIMD numeric types (dart:typed_data)**
- **Foreign function interfaces for interoperability with other code that presents a C-style interface (dartfish)**
- **Concurrent programming using *isolates*—independent workers that are similar to threads but don't share memory, communicating only through messages (dart: isolate)**
- **HTML elements and other resources for web-based applications that need to interact with the browser and the Document Object Model (DOM) (dart: html)**

Dart: The platforms

Dart's compiler technology lets you run code in different ways:

- **Native platform: For apps targeting mobile and desktop devices, Dart includes both a Dart VM with just-in-time (JIT) compilation and an ahead-of-time (AOT) compiler for producing machine code.**
- **Web platform: For apps targeting the web, Dart can compile for development or production purposes. Its web compiler translates Dart into JavaScript.**



User interface

- Introduction to widgets

Flutter widgets are built using a modern framework that takes inspiration from React. The central idea is that you build your UI out of widgets. Widgets describe what their view should look like given their current configuration and state. When a widget's state changes, the widget rebuilds its description, which the framework diffs against the previous description to

determine the minimal changes needed in the underlying render tree to transition from one state to the next.

Hello world.
The minimal Flutter app simply calls the runApp() function with a widget:

```dart
import 'package:flutter/material.dart';

void main() {
  runApp(
    const Center(
      child: Text(
        'Hello, world!',
        textDirection: TextDirection.ltr,
      ),
    ),
  );
```

The run App () function takes the given Widget and makes it the root of the widget tree. In this example, the widget tree consists of two widgets, the Center widget and its child, the Text widget. The framework forces the root widget to cover the screen, which means the text "Hello, world" ends up centered on screen. The text direction needs to be specified in this instance; when the Material App widget is used, this is taken care of for you, as demonstrated later.

When writing an app, you'll commonly author new widgets that are subclasses of either StatelessWidget or StatefulWidget, depending on whether your widget manages any state. A widget's main job is to

implement a build() function, which describes the widget in terms of other, lower-level widgets. The framework builds those widgets in turn until the process bottoms out in widgets that represent the underlying RenderObject, which computes and describes the geometry of the widget.

Basic widgets

Flutter comes with a suite of powerful basic widgets, of which the following are commonly used:

Text

The Text widget lets you create a run of styled text within your application.

Row, Column

These flex widgets let you create flexible layouts in both the horizontal (Row) and vertical (Column) directions. The design of these objects is based on the web's flexbox layout model.

Stack

Instead of being linearly oriented (either horizontally or vertically), a Stack widget lets you place widgets on top of each other in paint order. You can then use the Positioned widget on children of a Stack to position them relative to the top, right, bottom, or left edge of the stack. Stacks are based on the web's absolute positioning layout model.

Container:

The Container widget lets you create a rectangular visual element. A container can be decorated with a BoxDecoration, such as a background, a border, or a shadow. A Container can also have margins, padding, and constraints applied to its size. In addition, a Container can be transformed in three-dimensional space using a matrix.

Below are some simple widgets that combine these and other widgets:

```
import 'package: flutter/material. Dart';

class MyAppBar extends Stateless Widget {

  const MyAppBar ({required this. Title, super. Key});

  // Fields in a Widget subclass are always marked
"final".

  final Widget title;

  @override

  Widget build (Build Context context) {

    return Container (

      height: 56, // in logical pixels

      padding: const EdgeInsets.symmetric(horizontal: 8),

      decoration: Box Decoration (color: Colors. Blue
[500]),

      // Row is a horizontal, linear layout.
```

```
      child: Row (
        children: [
          const Icon Button (
            icon: Icon (Icons. Menu),
            tooltip: 'Navigation menu',
            on Pressed: null, // null disables the button
          ),
          // Expanded expands its child
          // to fill the available space.
          Expanded (
            child: title,
          ),
          const Icon Button (
            icon: Icon (Icons. Search),
            tooltip: 'Search',
            on Pressed: null,
          ),
        ],
      ),
    );
  }
}
```

```
class My Scaffold extends Stateless Widget {
  const My Scaffold ({super. Key});
  @override
  Widget build (Build Context context) {
    // Material is a conceptual piece
    // of paper on which the UI appears.
    return Material (
      // Column is a vertical, linear layout.
      child: Column (
        children: [
          MyAppBar(
            title: Text (
              'Example title',
              style: Theme. Of(context) //
                  .primaryTextTheme
                  . title Large,
            ),
          ),
          const Expanded (
            child: Center (
              child: Text ('Hello, world!'),
            ),
```

```
        ),
      ],
    ),
  );
 }
}
void main () {
 run App (
  const Material App (
    title: 'My app', // used by the OS task switcher
    home: Safe Area (
      child: My Scaffold (),
    ),
  ),
 );
}
```

Be sure to have a uses-material-design: true entry in the flutter section of your pubspec.yaml file. It allows you to use the predefined set of Material icons. It's generally a good idea to include this line if you are using the materials library.

```
name: my_app
flutter:
  uses-material-design: true
```

Many Material Design widgets need to be inside of a MaterialApp to display properly, in order to inherit theme data. Therefore, run the application with a Material App.

The MyAppBar widget creates a Container with a height of 56 device-independent pixels with an internal padding of 8 pixels, both on the left and the right. Inside the container, MyAppBar uses a Row layout to organize its children. The middle child, the title widget, is marked as Expanded, which means it expands to fill any remaining available space that hasn't been consumed by the other children. You can have multiple Expanded children and determine the ratio in which they consume the available space using the flex argument to Expanded.

The My Scaffold widget organizes its children in a vertical column. At the top of the column, it places an instance of MyAppBar, passing the app bar a Text widget to use as its title. Passing widgets as arguments to other widgets is a powerful technique that lets you create generic widgets that can be reused in a wide variety of ways.

Finally, My Scaffold uses an Expanded to fill the remaining space with its body, which consists of a centered message.

For more information, see Layouts.

Flutter logo

Standard lockup

The standard lockup can be used in slide decks and blog posts. It should never be used in-product or in a way that implies that Flutter is endorsing or has built the product.

Light

For lighter backgrounds with overlapping shapes.

Dark

For darker background with overlapping shapes or varied patterns when transparency is not available.

Transparency

For darker background backgrounds with overlapping shapes or varied patterns.

Knockout

For solid darker backgrounds with high contrasts

Logo variations

Vertical lockup

When there is limited horizontal space, a vertical lockup can be used

Logomark

When there is limited vertical and horizontal space, the logomark can be used by itself without the logotype.

Build with Flutter

When referring in the context of another app. This includes in blog posts, press interviews, and in the app itself.

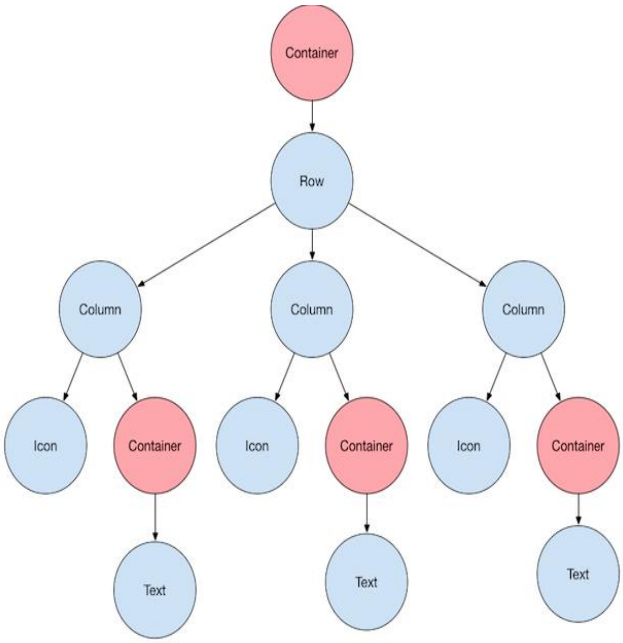
-Building Layout

Layouts in Flutter:

The core of Flutter's layout mechanism is widgets. In Flutter, almost everything is a widget—even layout models are widgets. The images, icons, and text that you see in a Flutter app are all widgets. But things you don't see are also widgets, such as the rows, columns, and grids that arrange, constrain, and align the visible widgets.

You create a layout by composing widgets to build more complex widgets. For example, the first screenshot below shows 3 icons with a label under each one:

Here's a diagram of the widget tree for this UI:



Most of this should look as you might expect, but you might be wondering about the containers (shown in pink). Container is a widget class that allows you to customize its child widget. Use a Container when you want to add padding, margins, borders, or background color, to name some of its capabilities.

In this example, each Text widget is placed in a Container to add margins. The entire Row is also placed in a Container to add padding around the row.

The rest of the UI in this example is controlled by properties. Set an Icon's color using its color property. Use the Text.style property to set the font, its color, weight, and so on. Columns and rows have properties that allow you to specify how their children are aligned vertically or horizontally, and how much space the children should occupy.

Material apps

For a Material app, you can use a Scaffold widget; it provides a default banner, background color, and has API for adding drawers, snack bars, and bottom sheets. Then you can add the Center widget directly to the body property for the home page.

lib/main.dart (MyApp)

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter layout demo',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Flutter layout demo'),
        ),
        body: const Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```

Non-Material apps

For a non-Material app, you can add the Center widget to the app's build () method.

lib/main.dart (MyApp)

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return Container(
      decoration: const BoxDecoration(color: Colors.white),
      child: const Center(
        child: Text(
          'Hello World',
          textDirection: TextDirection.ltr,
          style: TextStyle(
            fontSize: 32,
            color: Colors.black87,
          ),
        ),
      ),
    );
  }
}
```

Stateful and stateless widgets

A widget is either stateful or stateless. If a widget can change—when a user interacts with it, for example—it's stateful.

A *stateless* widget never changes. Icon, IconButton, and Text are examples of stateless widgets. Stateless widgets subclass StatelessWidget.

A *stateful* widget is dynamic: for example, it can change its appearance in response to events triggered by user interactions or when it receives data. Checkbox, Radio, Slider, InkWell, Form, and TextField are examples of stateful widgets. Stateful widgets subclass StatefulWidget.

A widget's state is stored in an State object, separating the widget's state from its appearance. The state consists of values that can change, like a slider's current value or whether a checkbox is checked. When the widget's state changes, the state object calls set State (), telling the framework to redraw the widget.

-Specifying assets

Flutter uses the pubspec.yaml file, located at the root of your project, to identify assets required by an app.

Here is an example:

```
flutter:
  assets:
    - assets/my_icon.png
    - assets/background.png
```

To include all assets under a directory, specify the directory name with the / character at the end:

```
flutter:
  assets:
    - directory/
    - directory/subdirectory/
```

Asset bundling

The assets subsection of the flutter section specifies files that should be included with the app. Each asset is identified by an explicit path (relative to the pubspec.yaml file) where the asset file is located. The

order in which the assets are declared doesn't matter. The actual directory name used (assets in first example or directory in the above example) doesn't matter.

During a build, Flutter places assets into a special archive called the *asset bundle* that apps read from at runtime.

- Animations

Introduction to animations

- **UI**
- **Animations**

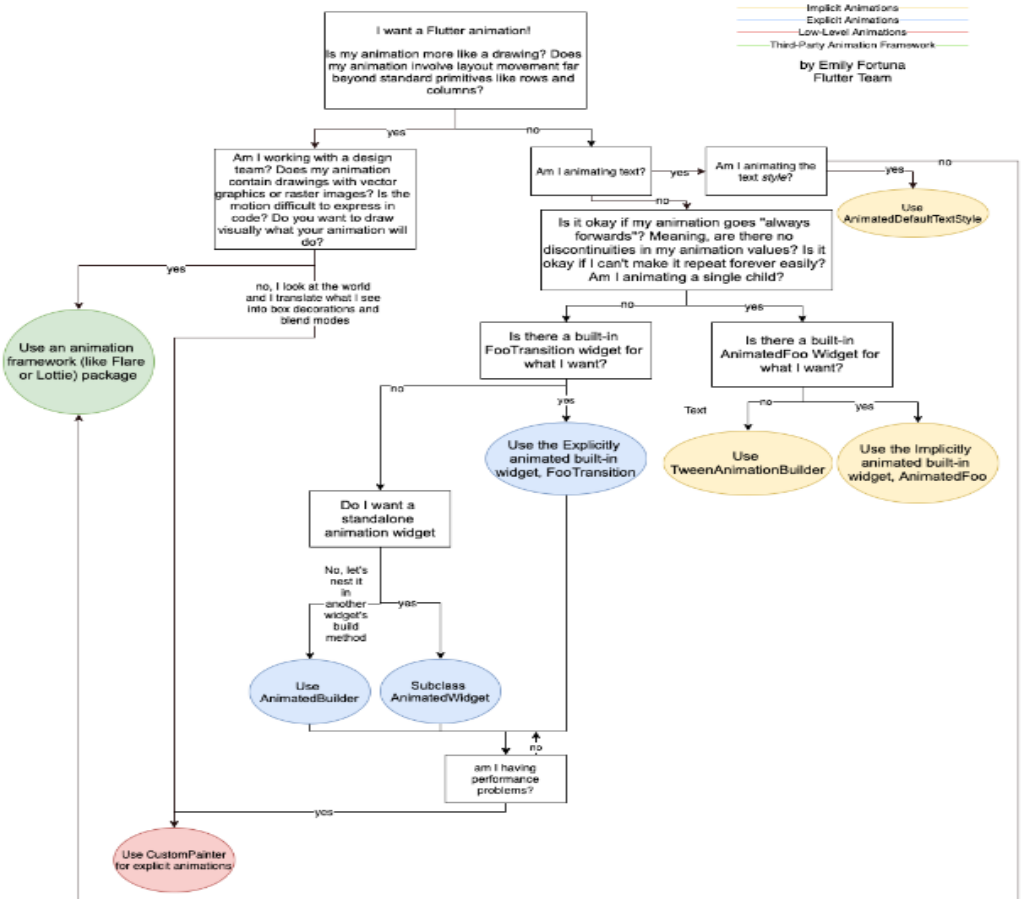
Well-designed animations make a UI feel more intuitive, contribute to the slick look and feel of a polished app, and improve the user experience. Flutter's animation support makes it easy to implement a variety of animation types. Many widgets, especially Material widgets, come with the standard motion effects defined in their design spec, but it's also possible to customize these effects.

Choosing an approach

There are different approaches you can take when creating animations in Flutter.

(To dive deeper into the decision process, watch the Animations in Flutter done right video, presented at Flutter Europe.)

As shown in the video, the following decision tree helps you decide what approach to use when implementing a Flutter animation:



Animation types

Generally, animations are either tween- or physics-based. The following sections explain what these terms mean, and point you to resources where you can learn more.

Tween animation

Short for *in-betweening*. In a tween animation, the beginning and ending points are defined, as well as a timeline, and a curve that defines the timing and speed of the transition. The framework calculates how to transition from the beginning point to the end point.

The documents listed above, such as the Animations tutorial, are not specifically about tweening, but they use tweens in their examples.

Physics-based animation

In physics-based animation, motion is modeled to resemble real-world behavior. When you toss a ball, for example, where and when it lands depends on how fast it was tossed and how far it was from the ground. Similarly, dropping a ball attached to a spring fall (and bounces) differently than dropping a ball attached to a string.

- **Animate a widget using a physics simulation**
  A recipe in the animations section of the Flutter cookbook.
- **Flutter Gallery**
  Under Material Components, the Grid example demonstrates a fling animation. Select one of the images from the grid and zoom in. You can pan the image with flinging or dragging gestures.
- **Also see the API documentation for AnimationController.animateWith and SpringSimulation.**

-Advanced UI

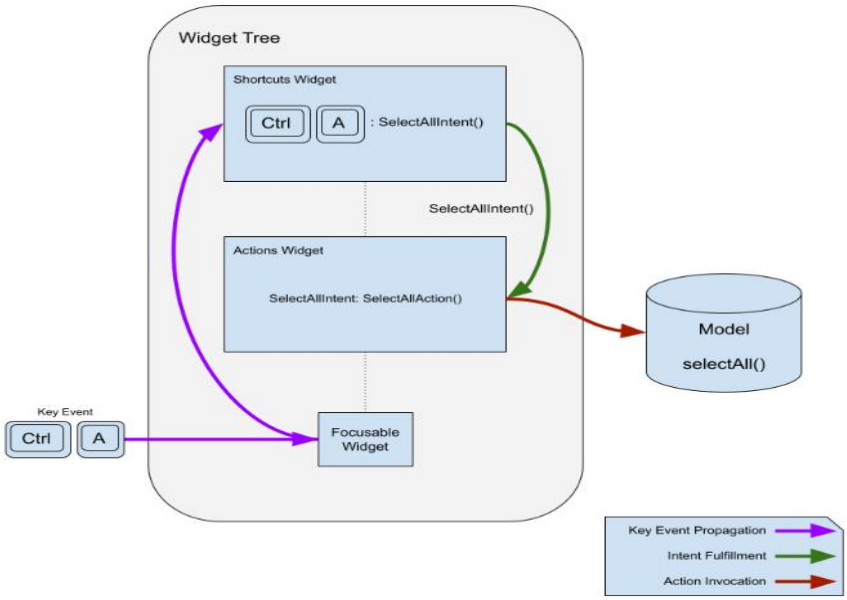
Using Actions and Shortcuts

Overview

For a GUI application to do anything, it has to have actions: users want to tell the application to *do* something. Actions are often simple functions that

directly perform the action (such as setting a value or save a file). In a larger application, however, things are more complex: the code for invoking the action, and the code for the action itself might need to be in different places. Shortcuts (key bindings) might need definition at a level that knows nothing about the actions they invoke.

That's where Flutter's actions and shortcuts system comes in. It allows developers to define actions that fulfill intents bound to them. In this context, an intent is a generic action that the user wishes to perform, and an Intent class instance represents these user intents in Flutter. An Intent can be general purpose, fulfilled by different actions in different contexts. An Action can be a simple callback (as in the case of the CallbackAction) or something more complex that integrates with entire undo/redo architectures (for example) or other logic.



Shortcuts are key bindings that activate by pressing a key or combination of keys. The key combinations reside in a table with their bound intent. When the Shortcuts widget invokes them, it sends their matching intent to the actions subsystem for fulfillment.

To illustrate the concepts in actions and shortcuts, this article creates a simple app that allows a user to select and copy text in a text field using both buttons and shortcut.

Features

-Hot reload
How to perform a hot reload

To hot reload a Flutter app:

1.Run the app from a supported Flutter editor or a terminal window. Either a physical or virtual device can be the target. Only Flutter apps in debug mode can be hot reloaded or hot restarted.

2.Modify one of the Dart files in your project. Most types of code changes can be hot reloaded; for a list of changes that require a hot restart, see Special cases.

3.If you're working in an IDE/editor that supports Flutter's IDE tools, select Save All (cmd-s/ctrl-s), or click the hot reload button on the toolbar.

If you're running the app at the command line using flutter run, enter r in the terminal window.

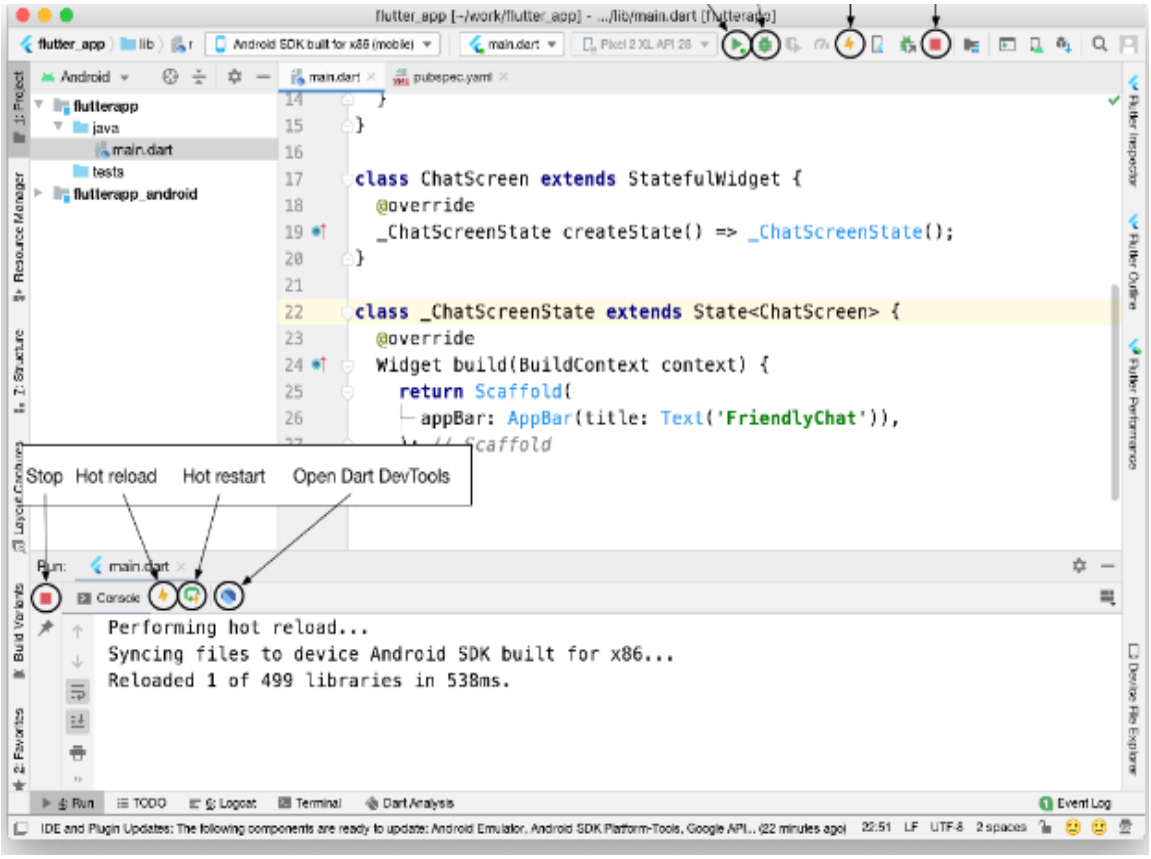
After a successful hot reload operation, you'll see a message in the console like:

Performing hot reload...
Reloaded 1 of 448 libraries in 978ms.

What is the difference between hot reload, hot restart, and full restart?

- **Hot reload loads code changes into the VM and re-builds the widget tree, preserving the app state; it doesn't rerun main () or initState(). (⌘\ in Intellij and Android Studio, ^F5 in VSCode)**
- **Hot restart loads code changes into the VM, and restarts the Flutter app, losing the app state. (⇧⌘\ in IntelliJ and Android Studio, ⇧⌘F5 in VSCode)**
- **Full restart restarts the iOS, Android, or web app. This takes longer because it also recompiles the Java / Kotlin / ObjC / Swift code. On the web, it also restarts the Dart Development Compiler. There is no specific keyboard shortcut for this; you need to stop and start the run configuration.**

***Flutter web currently supports hot restart but not hot reload.**



Navigation & Routing

Using the Navigator

The Navigator widget displays screens as a stack using the correct transition animations for the target platform. To navigate to a new screen, access the Navigator through the route's Build Context and call imperative methods such as push () or pop ():

The Navigator widget displays screens as a stack using the correct transition animations for the target platform. To navigate to a new screen, access the Navigator through the route's Build Context and call imperative methods such as push () or pop ():

```
onPressed: () {
  Navigator.of(context).push(
    MaterialPageRoute(
      builder: (context) => const SongScreen(song: song),
    ),
  );
},
child: Text(song.name),
```

Because Navigator keeps a stack of Route objects (representing the history stack), The push () method also takes a Route object. The MaterialPageRoute object is a subclass of Route that specifies the transition animations for Material Design. For more examples of how to use the Navigator, follow the navigation recipes from the Flutter Cookbook or visit the Navigator API documentation.

Using the Router

Flutter applications with advanced navigation and routing requirements (such as a web app that uses direct links to each screen, or an app with multiple Navigator widgets) should use a routing package such as go_router that can parse the route path and configure the Navigator whenever the app receives a new deep link.

To use the Router, switch to the router constructor on Material App or Cupertino App and provide it with a Router configuration. Routing packages, such as go_router, typically provide a configuration for you. For example:

```
MaterialApp.router(
  routerConfig: GoRouter(
    // …
  )
);
```

Because packages like go router are *declarative*, they will always display the same screen(s) when a deep link is received.

Data & Backend

Networking

- **Data & backend**
- **Networking**

Cross-platform http networking

The http package provides the simplest way to issue http requests. This package is supported on Android, iOS, macOS, Windows, Linux, and the web.

Platform notes

Some platforms require additional steps, as detailed below.

Android

Android apps must declare their use of the internet in the Android manifest (AndroidManifest.xml):

```
<manifest xmlns:android...>
 ...
 <uses-permission android:name="android.permission.INTERNET" />
 <application ...
</manifest>
```

macOS

macOS apps must allow network access in the relevant. entitlements files.

```
<key>com.apple.security.network.client</key>
<true/>
```

JSON and serialization

Which JSON serialization method is right for me?

This article covers two general strategies for working with JSON:

- **Manual serialization**
- **Automated serialization using code generation**

Different projects come with different complexities and use cases. For smaller proof-of-concept projects or quick prototypes, using code generators might be overkill. For apps with several JSON models with more complexity, encoding by hand can quickly become tedious, repetitive, and lend itself to many small errors.

Use manual serialization for smaller projects.

Manual JSON decoding refers to using the built-in JSON decoder in dart: convert. It involves passing the raw JSON string to the Json Decode () function, and then looking up the values you need in the resulting Map<String, dynamic>. It has no external dependencies or particular setup process, and it's good for a quick proof of concept.

Manual decoding does not perform well when your project becomes bigger. Writing decoding logic by hand can become hard to manage and error-prone. If you have a typo when accessing a nonexistent JSON field, your code throws an error during runtime.

If you do not have many JSON models in your project and are looking to test a concept quickly, manual serialization might be the way you want to start. For an example of manual encoding, see Serializing JSON manually using dart:convert.

Use code generation for medium to large projects. JSON serialization with code generation means having an external library generate the encoding boilerplate for you. After some initial setup, you run a file watcher that generates the code from your model classes. For example, json_serializable and built_value are these kinds of libraries.

This approach scales well for a larger project. No hand-written boilerplate is needed, and typos when accessing JSON fields are caught at compile-time. The downside with code generation is that it requires some initial setup. Also, the generated source files might produce visual clutter in your project navigator.

You might want to use generated code for JSON serialization when you have a medium or a larger project. To see an example of code generation-based JSON encoding, see Serializing JSON using code generation libraries

Accessibility & localization

Ensuring apps are accessible to a broad range of users is an essential part of building a high-quality app.

Applications that are poorly designed create barriers to people of all ages. The UN Convention on the Rights of Persons with Disabilities states the moral and legal imperative to ensure universal access to information systems; countries around the world enforce accessibility as a requirement; and companies recognize the business advantages of maximizing access to their services.

We strongly encourage you to include an accessibility checklist as a key criterion before shipping your app. Flutter is committed to supporting developers in making their apps more accessible, and includes first-class framework support for accessibility in addition to that provided by the underlying operating system, including:

Large fonts

Render text widgets with user-specified font sizes

Screen readers

Communicate spoken feedback about UI contents

Sufficient contrast

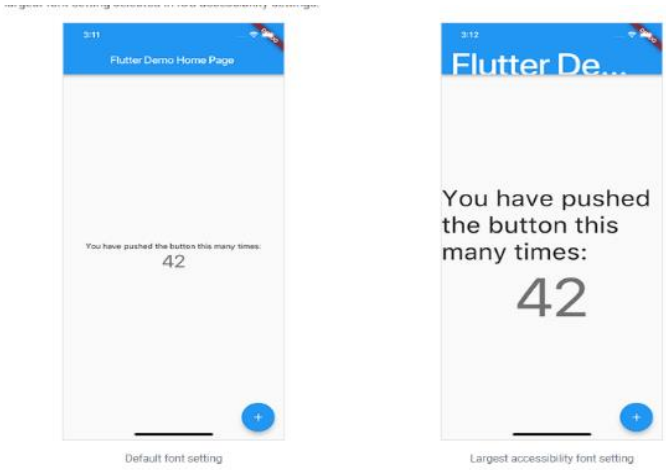
Render widgets with colors that have sufficient contrast.

Large fonts
Both Android and iOS contain system settings to configure the desired font sizes used by apps. Flutter

text widgets respect this OS setting when determining font sizes.

Font sizes are calculated automatically by Flutter based on the OS setting. However, as a developer you should make sure your layout has enough room to render all its contents when the font sizes are increased. For example, you can test all parts of your app on a small-screen device configured to use the largest font setting.

Example:
The following two screenshots show the standard Flutter app template rendered with the default iOS font setting, and with the largest font setting selected in iOS accessibility settings.



-Introduction to localizations in Flutter

This section provides a tutorial on how to create and internationalize a new Flutter application, along with any additional setup that a target platform might require.

You can find the source code for this example in gen_l10n_example.

Setting up an internation-alized app: the Flutter localizations package

By default, Flutter only provides US English localizations. To add support for other languages, an application must specify additional Material App (or Cupertino App) properties, and include a package called flutter localizations. As of June 2023, this package supports 113 languages and language variants.

To begin, start by creating a new Flutter application in a directory of your choice with the flutter create command.

```
$ flutter create <name_of_flutter_app>
```

To use `flutter_localizations`, add the package as a dependency to your `pubspec.yaml` file, as well as the `intl` package:

```
$ flutter pub add flutter_localizations --sdk=flutter
$ flutter pub add intl:any
```

This creates a `pubspec.yml` file with the following entries:

```
dependencies:
  flutter:
    sdk: flutter
  flutter_localizations:
    sdk: flutter
  intl: any
```

Then import the `flutter_localizations` library and specify `localizationsDelegates` and `supportedLocales` for your `MaterialApp` or `CupertinoApp`:

```
import 'package:flutter_localizations/flutter_localizations.dart';
```

```
return const MaterialApp(
  title: 'Localizations Sample App',
  localizationsDelegates: [
    GlobalMaterialLocalizations.delegate,
    GlobalWidgetsLocalizations.delegate,
    GlobalCupertinoLocalizations.delegate,
  ],
  supportedLocales: [
    Locale('en'), // English
    Locale('es'), // Spanish
  ],
  home: MyHomePage(),
);
```

After introducing the flutter localizations package and adding the previous code, the Material and Cupertino packages should now be correctly localized in one of the 113 supported locales. Widgets should be adapted

to the localized messages, along with correct left-to-right or right-to-left layout.

Try switching the target platform's locale to Spanish (es) and the messages should be localized.

Apps based on Widgets App are similar except that the GlobalMaterialLocalizations.delegate isn't needed.

The full Locale.fromSubtags constructor is preferred as it supports script Code, though the Locale default constructor is still fully valid.

The elements of the localizations Delegates list are factories that produce collections of localized values. GlobalMaterialLocalizations.delegate provides localized strings and other values for the Material Components library. GlobalWidgetsLocalizations.delegate defines the default text direction, either left-to-right or right-to-left, for the widget's library.

More information about these app properties, the types they depend on, and how internationalized Flutter apps are typically structured, is covered in this page.

Packages & Plugins

-Developing packages & plugins
Package introduction

Packages enable the creation of modular code that can be shared easily. A minimal package consists of the following:

pubspec.yaml

A metadata file that declares the package name, version, author, and so on.

lib

The lib directory contains the public code in the package, minimally a single <package-name>.dart file. Package types.

Packages can contain more than one kind of content:

Dart packages

General packages written in Dart, for example the path package. Some of these might contain Flutter specific functionality and thus have a dependency on the Flutter framework, restricting their use to Flutter only, for example the fluro package.

Plugin packages

A specialized Dart package that contains an API written in Dart code combined with one or more platform-specific implementations.

Plugin packages can be written for Android (using Kotlin or Java), iOS (using Swift or Objective-C), web,

macOS, Windows, or Linux, or any combination thereof.

A concrete example is the url_launcher plugin package.
Debugging Flutter apps

Testing & Debugging

There's a wide variety of tools and features to help debug Flutter applications. Here are some of the available tools:

- **DevTools, a suite of performance and profiling tools that run in a browser.**
- **Android Studio/IntelliJ, and VS Code (enabled with the Flutter and Dart plugins) support a built-in source-level debugger with the ability to set breakpoints, step through code, and examine values.**
- **Flutter inspector, a widget inspector available in DevTools, and directly from Android Studio and IntelliJ (enabled with the Flutter plugin). The inspector allows you to examine a visual representation of the widget tree, inspect individual widgets and their property values, enable the performance overlay, and more.**
- **If you are looking for a way to use GDB to remotely debug the Flutter engine running within an Android app process, check out flutter_gdb.**

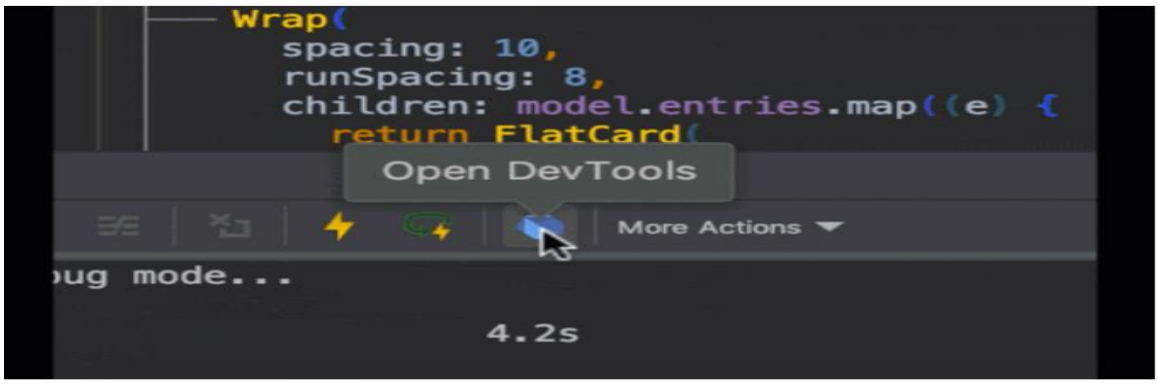
DevTools

For debugging and profiling apps, DevTools might be the first tool you reach for. DevTools runs in a browser and supports a variety of features:

- **source-level debugger**
- **widget inspector that displays a visual widget tree, and "widget select" mode where you select a widget in the app, and it drills down to that widget in the tree**
- **memory profiler**
- **timeline view that supports tracing, and importing and exporting trace information**
- **logging view**

If you run your application in debug mode or profile mode, while it's running you can open DevTools in the browser to connect to your app. DevTools doesn't work well with an app compiled to release mode, as the debugging and profiling information has been stripped away.

If you use DevTools for profiling, make sure to run your application in profile mode. Otherwise, the main output that appears on your profile are the debug asserts verifying the framework's various invariants (see Debug mode assertions).



-Testing plugins

Types of plugin tests

To see examples of each of these types of tests, you can create a new plugin from the plugin template and look in the indicated directories.

- **Dart unit tests and widget tests. These tests allow you to test the Dart portion of your plugin just as you would test the Dart code of a non-plugin package. However, the plugin's native code won't be loaded, so any calls to platform channels need to be mocked in tests.**
- **See the test directory for an example.**
- **Dart integration tests. Since integration tests run in the context of a Flutter application (the example app), they can test both the Dart and native code, as well as the interaction between them. They are also useful for unit testing web implementation code that needs to run in a browser.**
- **These are often the most important tests for a plugin. However, Dart integration tests can't**

interact with native UI, such as native dialogs or the contents of platform views.

- **See the example/integration test directory for an example.**
- **Native unit tests. Just as Dart unit tests can test the Dart portions of a plugin in isolation, native unit tests can test the native parts in isolation. Each platform has its own native unit test system, and the tests are written in the same native languages as the code it is testing.**
- **Native unit tests can be especially valuable if you need to mock out APIs wrapped by your plugin code, which isn't possible in a Dart integration test.**
- **You can set up and use any native test frameworks you are familiar with for each platform, but the following are already configured in the plugin template:**
    - **Android: JUnit tests can be found in android/src/test/.**
    - **iOS and macOS: XCTest tests can be found in example/iOS/RunnerTests/ and example/macOS/RunnerTests/ respectively. These are in the example directory, not the top-level package directory, because they are run via the example app's project.**

- **Linux and Windows: Google_Test tests can be found in Linux/test/ and windows/test/, respectively.**

Handling errors in Flutter

Errors caught by Flutter.

```
import 'dart:io';

import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';

void main() {
  FlutterError.onError = (details) {
    FlutterError.presentError(details);
    if (kReleaseMode) exit(1);
  };
  runApp(const MyApp());
}

// rest of `flutter create` code...
```

Testing Flutter apps

Automated testing falls into a few categories:

- **A *unit test* tests a single function, method, or class.**
- **A *widget test* (in other UI frameworks referred to as *component test*) tests a single widget.**
- **An *integration test* tests a complete app or a large part of an app.**

Generally speaking, a well-tested app has many unit and widget tests, tracked by code coverage, plus enough integration tests to cover all the important use cases. This advice is because there are trade-offs between different kinds of testing, seen below.

	Unit	Widget	Integration
Confidence	Low	Higher	Highest
Maintenance cost	Low	Higher	Highest
Dependencies	Few	More	Most
Execution speed	Quick	Quick	Slow

Unit tests

A *unit test* tests a single function, method, or class. The goal of a unit test is to verify the correctness of a unit of logic under a variety of conditions. External dependencies of the unit under test are generally mocked out. Unit tests generally don't read from or write to disk, render to screen, or receive user actions from outside the process running the test. For more information regarding unit tests, you can view the following recipes or run flutter test --help in your terminal.

Widget tests

A *widget test* (in other UI frameworks referred to as *component test*) tests a single widget. The goal of a widget test is to verify that the widget's UI looks and interacts as expected. Testing a widget involves multiple classes and requires a test environment that provides the appropriate widget lifecycle context.

For example, the Widget being tested should be able to receive and respond to user actions and events, perform layout, and instantiate child widgets. A widget

test is therefore more comprehensive than a unit test. However, like a unit test, a widget test's environment is replaced with an implementation much simpler than a full-blown UI system.

Integration tests,
An *integration test* tests a complete app or a large part of an app. The goal of an integration test is to verify that all the widgets and services being tested work together as expected. Furthermore, you can use integration tests to verify your app's performance.

Generally, an *integration test* runs on a real device or an OS emulator, such as iOS Simulator or Android Emulator. The app under test is typically isolated from the test driver code to avoid skewing the results.

For more information on how to write integration tests, see the integration testing page.

Performance & Optimization

Measuring your app's size
Debug builds are not representative.
By default, launching your app with flutter run, or by clicking the Play button in your IDE (as used in Test drive and Write your first Flutter app), generates a *debug* build of the Flutter app. The app size of a debug build is large due to the debugging overhead that allows for hot reload and source-level debugging. As

such, it is not representative of a production app end users download.

Checking the total size

A default release build, such as one created by flutter build apk or flutter build iOS is built to conveniently assemble your upload package to the Play Store and App Store. As such, they're also not representative of your end-users' download size. The stores generally reprocess and split your upload package to target the specific downloader and the downloader's hardware, such as filtering for assets targeting the phone's DPI, filtering native libraries targeting the phone's CPU architecture.

Estimating total size

To get the closest approximate size on each platform, use the following instructions.

Android

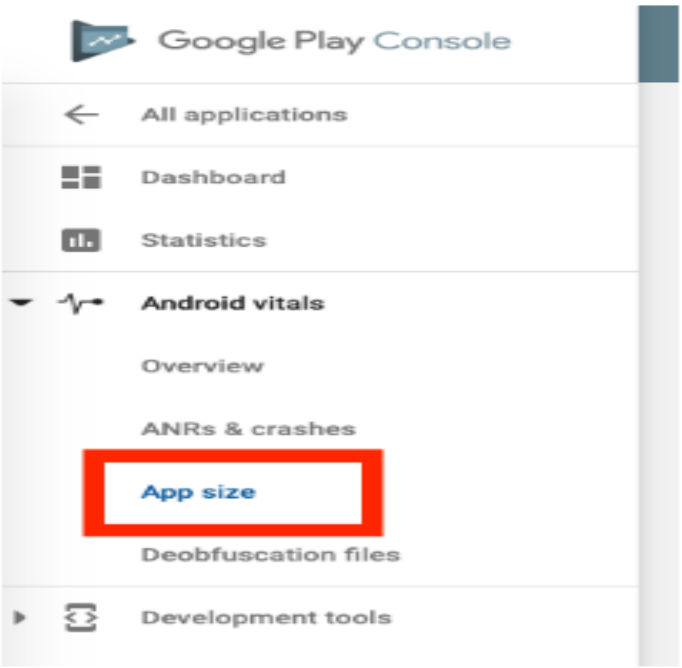
Follow the Google Play Console's instructions for checking app download and install sizes.

Produce an upload package for your application:

```
flutter build appbundle
```

Log into your Google Play Console. Upload your application binary by drag dropping the .aab file.

View the application's download and install size in the Android vitals -> App size tab.



The download size is calculated based on an XXXHDPI (~640dpi) device on an arm64-v8a architecture. Your end users' download sizes may vary depending on their hardware.

The top tab has a toggle for download size and install size. The page also contains optimization tips further below.

Deployment

-Create flavors of a Flutter app

What are flavors?
Have you ever wondered how to set up different environments in your Flutter app? Flavors (known as *build configurations* in iOS), allow you (the developer) to create separate environments for your app using the same code base. For example, you might have one flavor for your full-fledged production app, another as

a limited “free” app, another for testing experimental features, and so on.

Say you want to make both free and paid versions of your Flutter app. You can use flavors to set up both app versions without writing two separate apps. For example, the free version of the app has basic functionality and ads. In contrast, the paid version has basic app functionality, extra features, different styles for paid users, and no ads.

You also might use flavors for feature development. If you’ve built a new feature and want to try it out, you could set up a flavor to test it out. Your production code remains unaffected until you’re ready to deploy your new feature.

Flavors let you define compile-time configurations and set parameters that are read at runtime to customize your app’s behavior.

This document guides you through setting up Flutter flavors for iOS and Android.

Environment set up.
Prerequisites:

- **Xcode installed.**
- **An existing Flutter projects.**

To set up flavors in iOS, you’ll define build configurations in Xcode.

Creating flavors in iOS

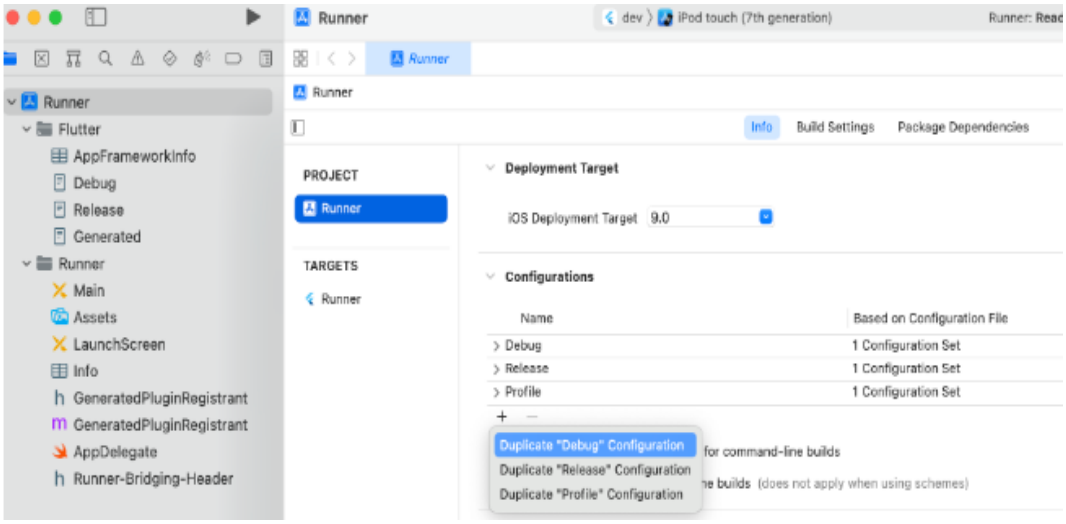
1.Open your project in Xcode.

2.Select Product > Scheme > New Scheme from the menu to add a new Scheme.

- **A scheme describes how Xcode runs different actions. For the purposes of this guide, the example *flavor* and *scheme* are named free. The build configurations in the free scheme have the -free suffix.**

3.Duplicate the build configurations to differentiate between the default configurations that are already available and the new configurations for the free scheme.

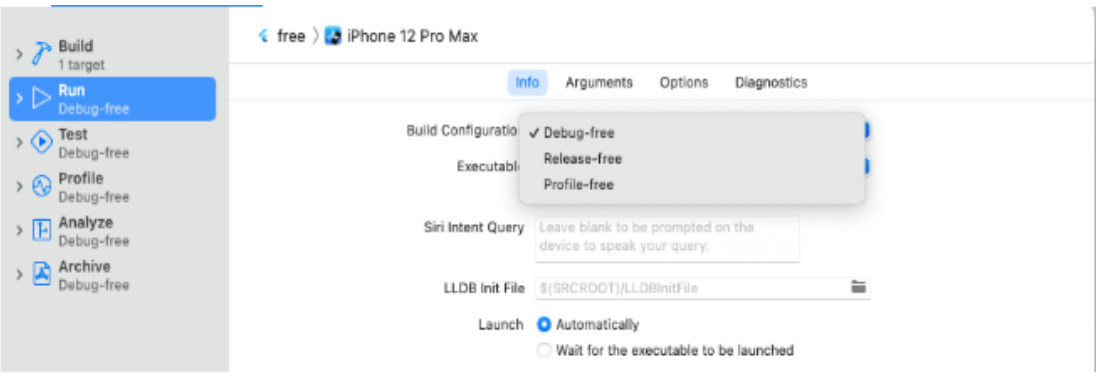
- **Under the Info tab at the end of the Configurations dropdown list, click the plus button and duplicate each configuration name (Debug, Release, and Profile). Duplicate the existing configurations, once for each environment.**



4.To match the free flavor, add -free at the end of each new configuration name.

5.Change the free scheme to match the build configurations already created.

- **In the Runner project, click Manage Schemes… and a pop-up window opens.**
- **Double click the free scheme. In the next step (as shown in the screenshot), you'll modify each scheme to match its free build configuration:**



Firebase

Email Authentication and Password Reset in Flutter

Introduction:

This documentation provides a step-by-step guide on how to implement email authentication and password reset using Firebase in a Flutter application. Firebase is a popular mobile and web development platform that offers a variety of services, including authentication. With Firebase Authentication, developers can easily add user authentication to their apps and authenticate users using their email addresses and passwords.

In this guide, we'll cover the following topics:

1 - Setting up a Flutter project with Firebase.

2 - Configuring Firebase Authentication.

3 - Implementing email authentication.

4 - Implementing password reset functionality.

Prerequisites

Before proceeding with this guide, make sure you have the following prerequisites:

1 - Flutter SDK installed on your system.

2 - Flutter project created using flutter create.

3 - Firebase project created in the Firebase console

(https://console.firebase.google.com).

Step 1: Setting up a Flutter project with Firebase

1 - Open your Flutter project in your preferred editor.

2 - Add the firebase_core and firebase_auth dependencies to your pubspec.yaml file:

```
dependencies:
  flutter:
    sdk: flutter
  firebase_core: ^1.10.0
  firebase_auth: ^3.3.0
```

3 - Run flutter pub get to fetch the dependencies.

4 - Navigate to the Firebase console

(https://console.firebase.google.com) and create a new project.

5 - Click on "Add app" and select the Flutter icon to register your app with Firebase.

6 - Follow the instructions to add the google-services.json file to your Flutter project.

7 - Initialize Firebase in your Flutter app by adding the following code to your main.dart file:

```
import 'package:firebase_core/firebase_core.dart';

void main() async {

  WidgetsFlutterBinding.ensureInitialized();

  await Firebase.initializeApp();

  runApp(MyApp());

}
```

Make sure to replace MyApp with your actual app widget.

Step 2: Configuring Firebase Authentication

1 - Open the Firebase console and navigate to your project.

2 - Click on "Authentication" in the left sidebar.

3 - Enable the "Email/Password" sign-in method.

Step 3: Implementing email authentication.

Create a new Dart file (e.g., auth_service.dart) to handle authentication-related functionality.

Import the necessary packages:

```
import 'package:firebase_auth/firebase_auth.dart';
```

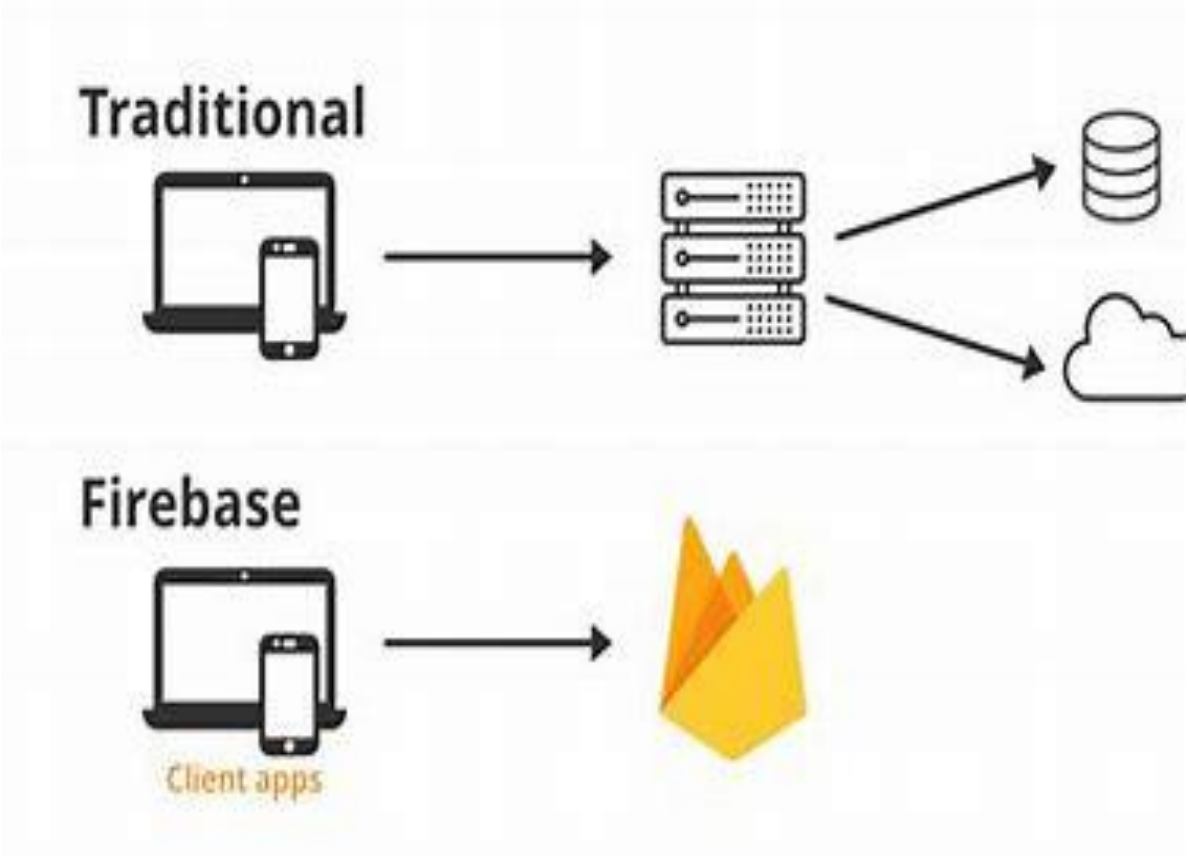
3 -Create a class named AuthService and add the following methods:

```
class AuthService {

  final FirebaseAuth _auth = FirebaseAuth.instance;

  Future<UserCredential?>
signUpWithEmailAndPassword(

     String email, String password) async {

   try {

     final UserCredential userCredential = await _auth

        .createUserWithEmailAndPassword(email: email,
password: password);

     return userCredential;
```

```
    } catch (e) {
      print('Error during email/password signup: $e');
      return null;
    }
  }
  Future<UserCredential?>
signInWithEmailAndPassword(
      String email, String password) async {
    try {
      final UserCredential userCredential = await _auth
          .signInWithEmailAndPassword(email: email,
password: password);
      return userCredential;
    } catch (e) {
      print('Error during email/password sign-in: $e');
      return null;
    }
  }
  Future<void> signOut() async {
```

```
    await _auth.signOut();
  }
}
```

4 - Now, you can use the AuthService class to handle email authentication in your app.



The Application

Sign in

First, when the user enters the Application, he should sign up for the first time and after that he can sign in using the email and password only in the sign in page.

Sign in page format is below: -

Sign up.

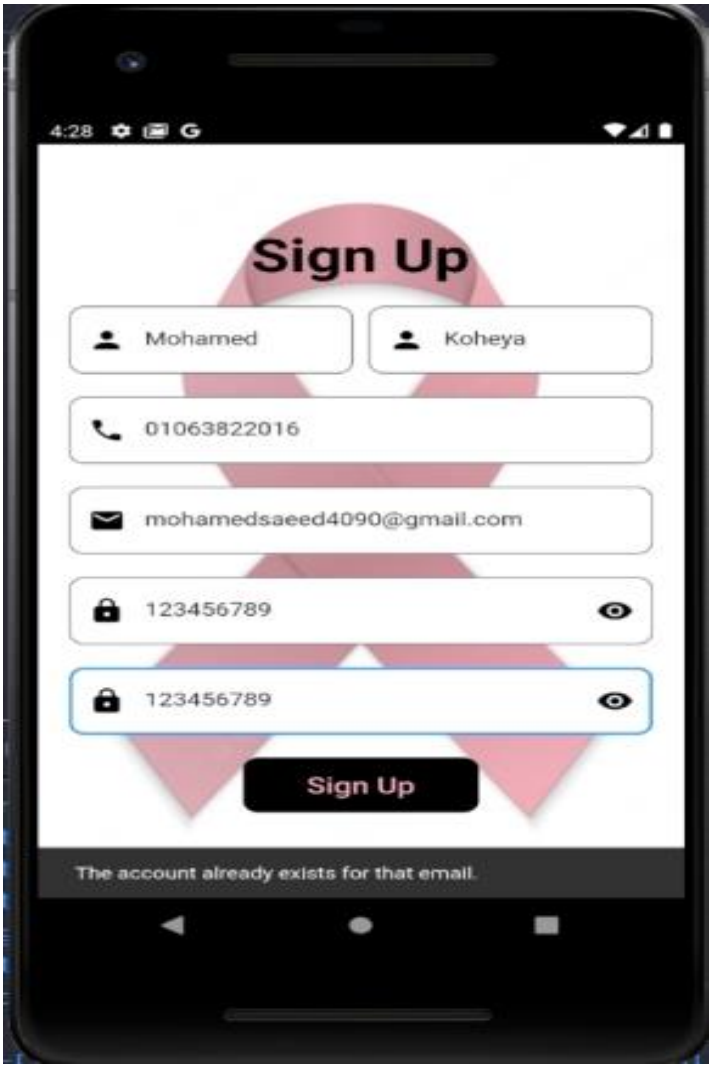
If the user didn't sign up before, the application asks the user to enter some information such as name, phone number and password so that the user can create a signup.

In this page, there is a text form field, meaning that these fields must not be empty.

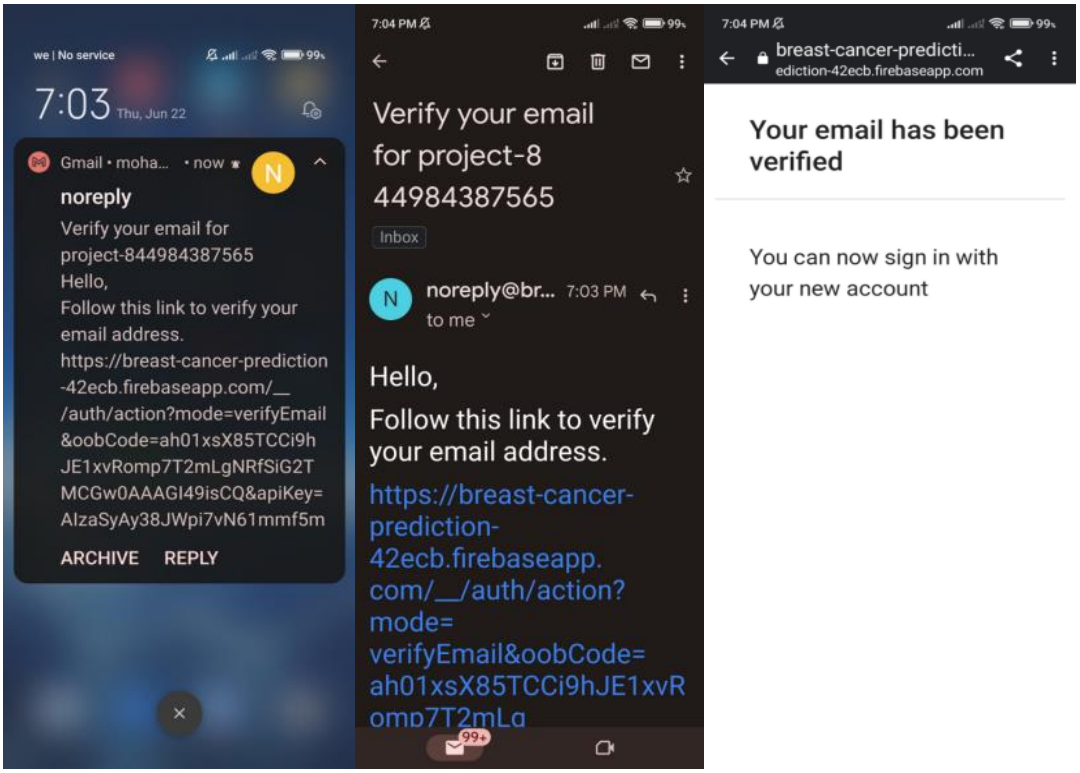
In this picture, he shows that he likes the first password that matches the password you entered before. so that a message does not appear that it is a wrong password, as in the picture.

***All pictures we used here are from our real application.**

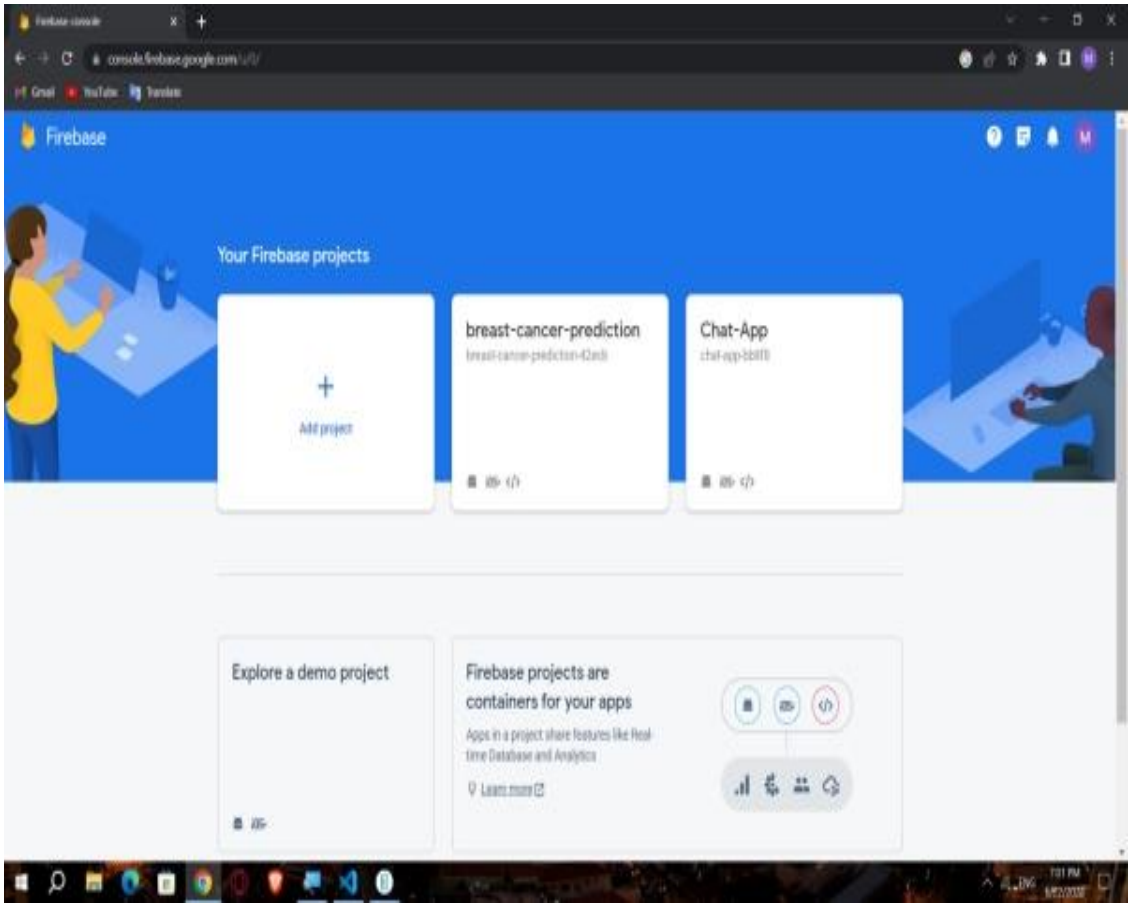
If the user enters an email that was previously registered, a message appears that this email already exists.

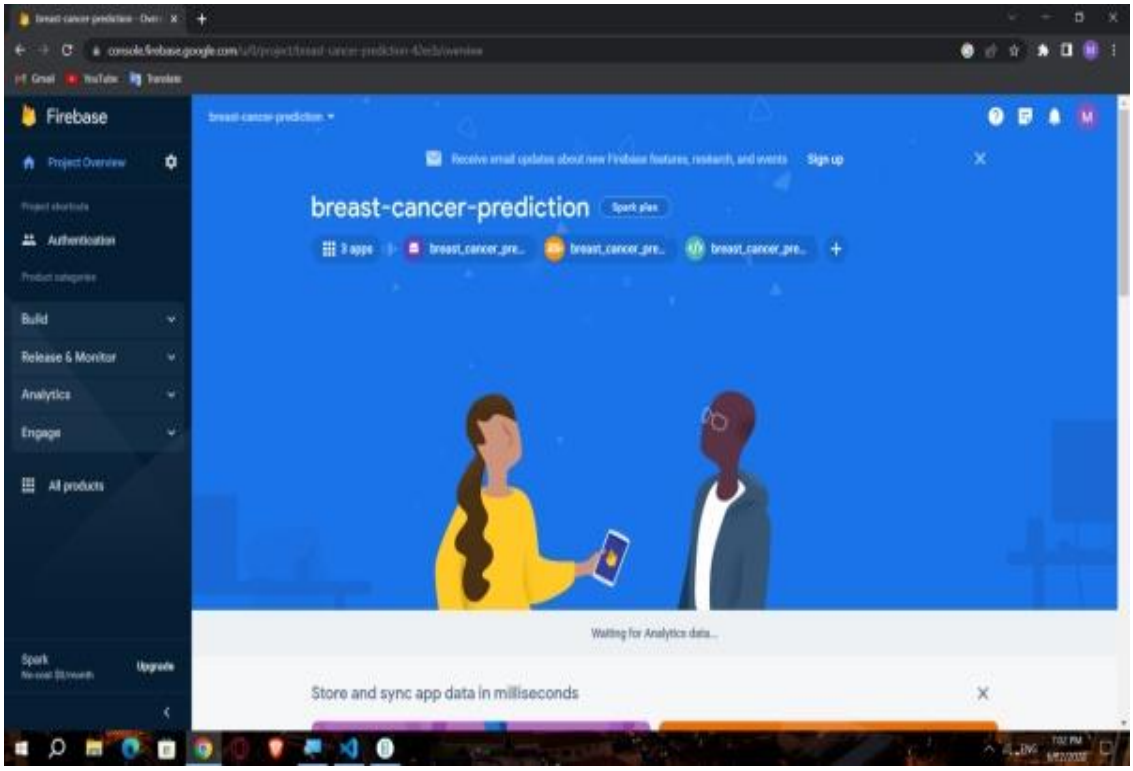


The user receives a message Until it verifies that the email entered is correct and the user opens it, then this message appears as in the last picture.

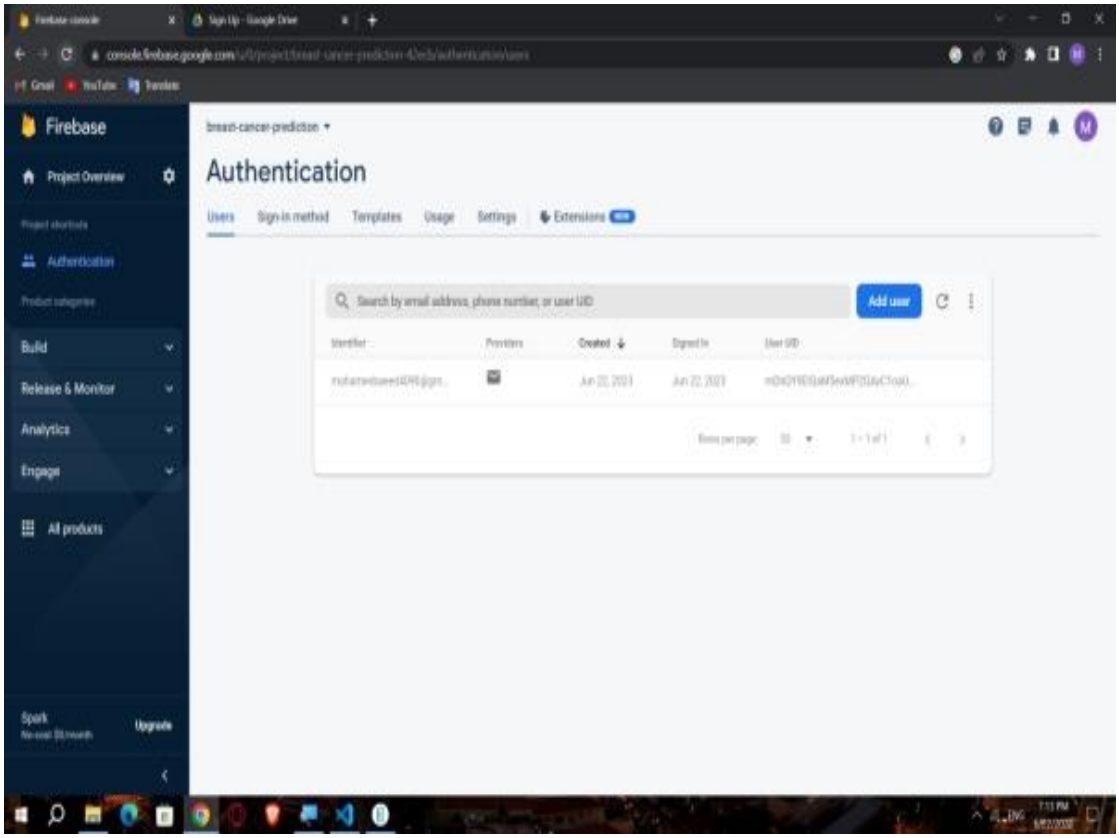


I choose the name of my project from firebase.





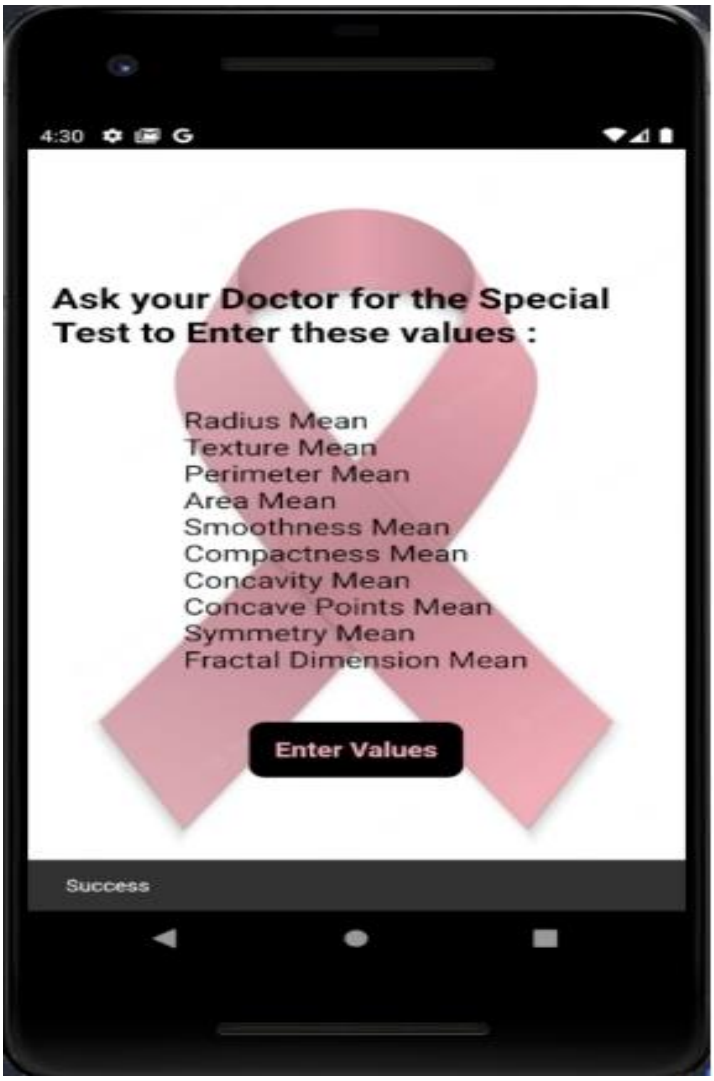
the names of the users who have registered before appearing.



Home Page

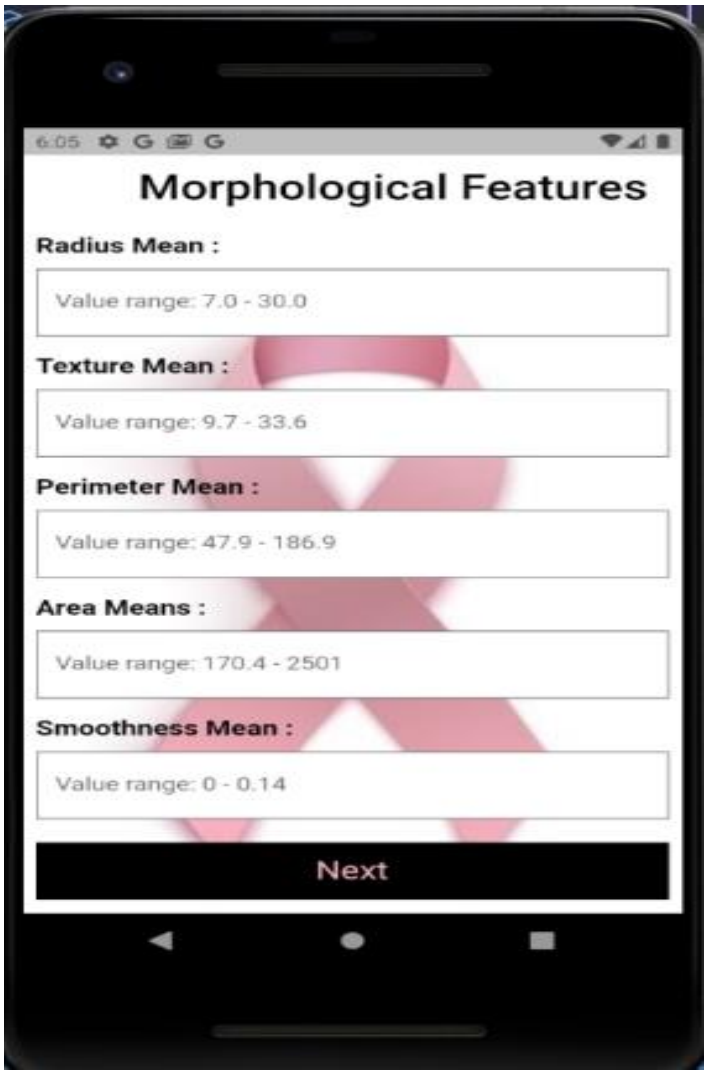
In this page:

The patient must know these values (Radius Mean, Texture Mean, Perimeter Mean, Area Mean, Smoothness Mean, Compactness Mean, Concavity Mean, Concave Points Mean, Symmetry Mean, Fractal Dimension Mean) from the doctor to be entered to know the Result. Once we press the Enter Values button, we will move the values page to be entered.



Morphological Features

In this page, the application asks the user to enter the first of 5 values with a specific range for each value, as shown in the image.

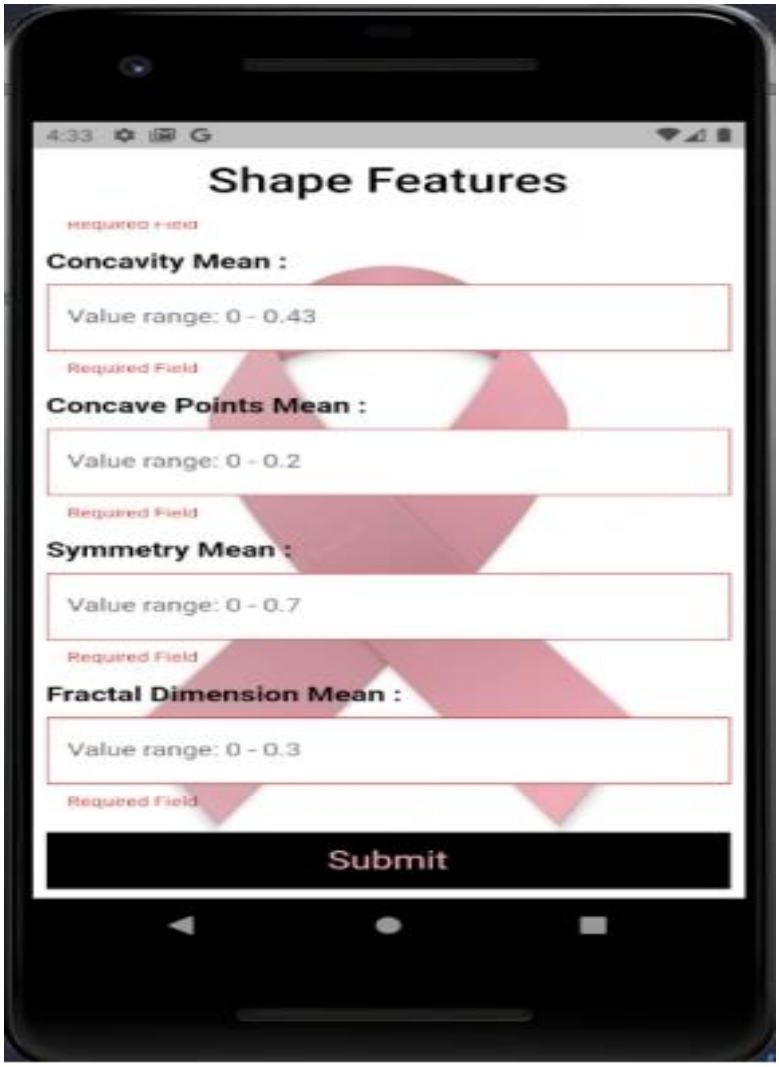


If the user did not enter values in each field, a message will appear under the field in which the values were not entered. It is obligatory for the user to enter this value as shown in the images.

If the user entered a value outside the limits of the range. A message will appear that the value is outside the ring, as shown in the picture.

Shape Features

On this page, the application asks the user to enter the rest of the values with a specific range for each value, as shown in the image.



Result

The application processes the data to determine whether the person is Benign or malignant and shows the result as shown in the picture.

On this page there are 4 buttons

1-Benign button

In the case of the result appears to the person that the tumor is benign, the person presses this button.

2- Malignant button

In the case of the result appears to the person that the tumor is malignant, the person presses this button.

3-About Breast Cancer button

The person presses on this button to know more about breast cancer.

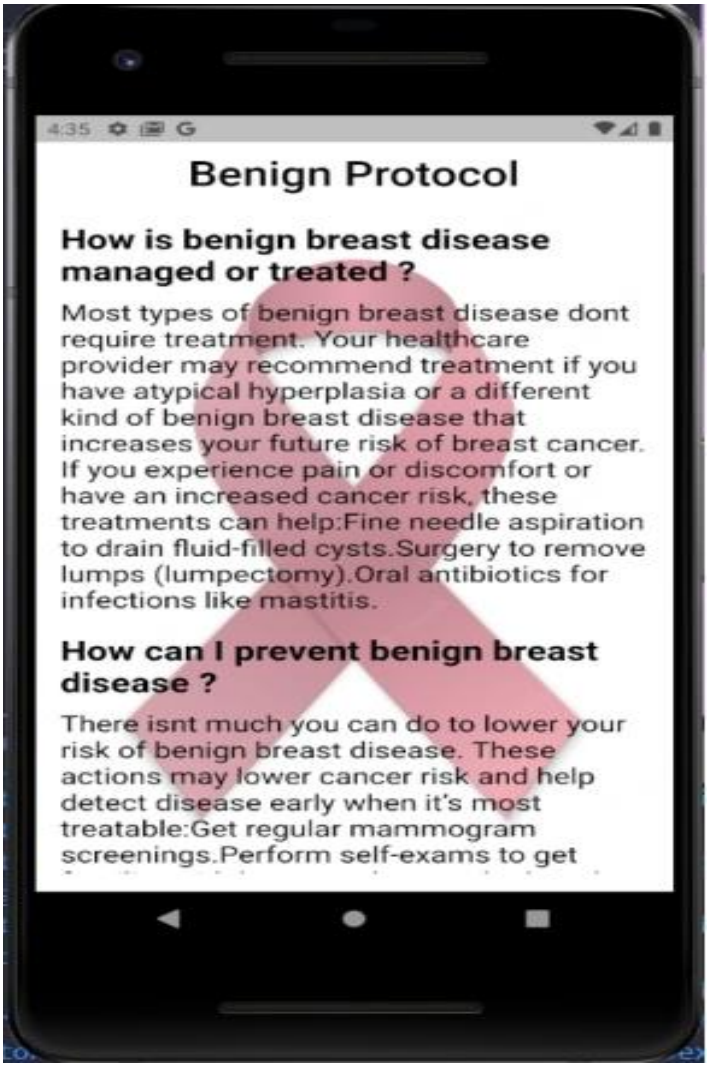
4-Re Enter Values button

The person presses on this button to enter the values again that were entered before to confirm the result that appeared.

Benign Protocol

This page appears in the case of the person pressing the Benign button.

The page contains the Benign protocol, such as how to treat this type of tumor and how to prevent breast cancer, and so on, as shown in the picture.



Malignant Protocol

This page appears in the case of the person presses on the Malignant button.

The page contains the Malignant protocol, for example, the type of drug should I follow
What are the stages of this type of tumor and how is each stage treated?

About Breast Cancer

This page appears in the case of the person presses on the About Breast Cancer button.

The page contains information about breast cancer, for example, what it is and its types, and so on, as shown in the picture.

What is breast Cancer ?

Breast cancer is a type of cancer that develops in the breast tissue. It is one of the most common types of cancer among women, although men can also develop breast cancer.
Breast cancer can begin in different parts of the breast, such as the ducts that carry milk to the nipple or the glands that produce milk. It can also spread to other parts of the body, such as the lymph nodes, bones, liver, or lungs.

Types of Breast Cancer ?

The type of breast cancer is determined by the characteristics of the cancer cells, such as how they look under a microscope and whether they have specific receptors or proteins.The type of breast cancer can affect the treatment plan and prognosis.

1. Ductal Carcinoma In Situ (DCIS): This is a non-invasive type of breast cancer that starts in the milk ducts and has not spread to the surrounding breast tissue.

2. Invasive Ductal Carcinoma (IDC): This is the most common type of breast cancer, which starts in the milk ducts and invades the surrounding breast tissue.

3. Invasive Lobular Carcinoma (ILC): This type of breast cancer starts in the milk-producing glands (lobules) and can spread to other parts of the breast and beyond.

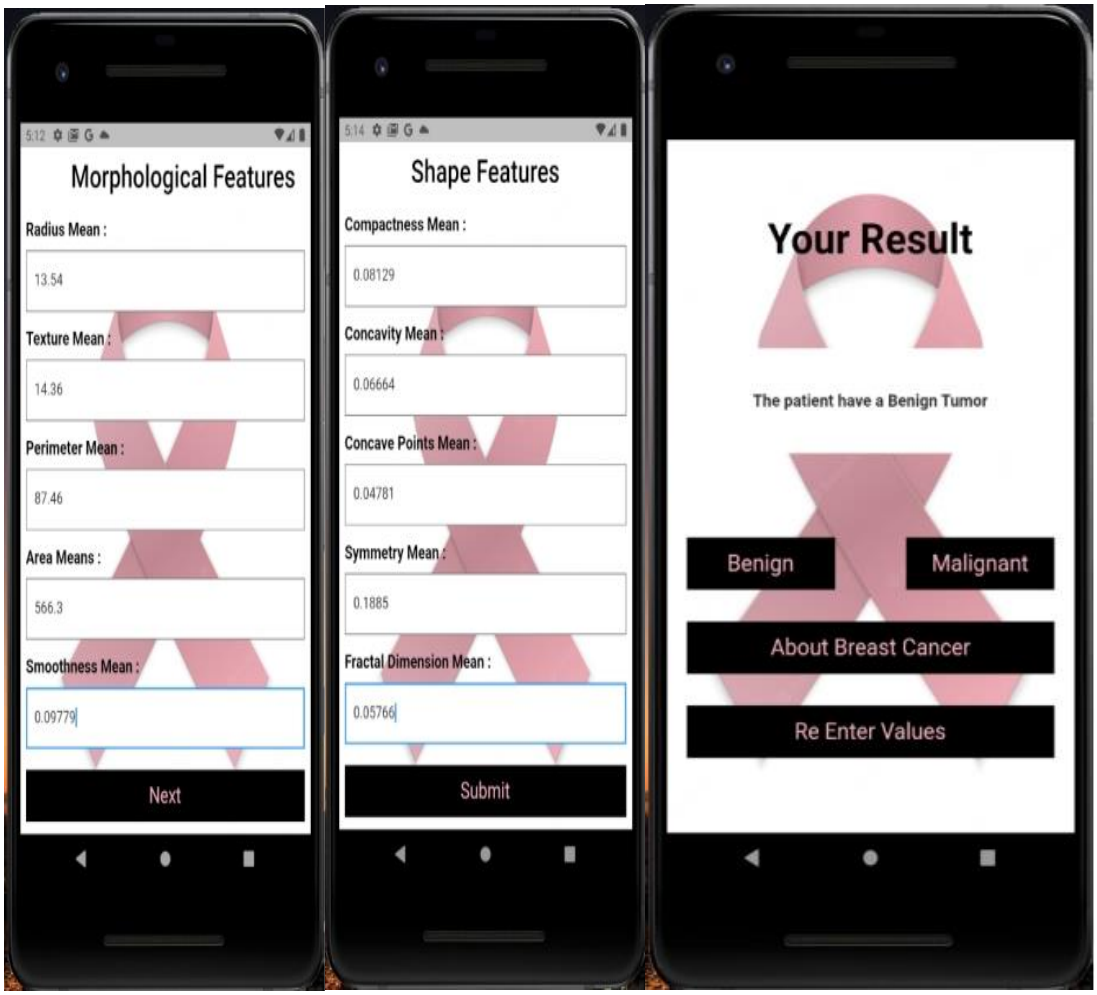
4. Inflammatory Breast Cancer (IBC): This is a rare but aggressive type of breast cancer that causes the breast to become red, swollen, and warm.

Testing

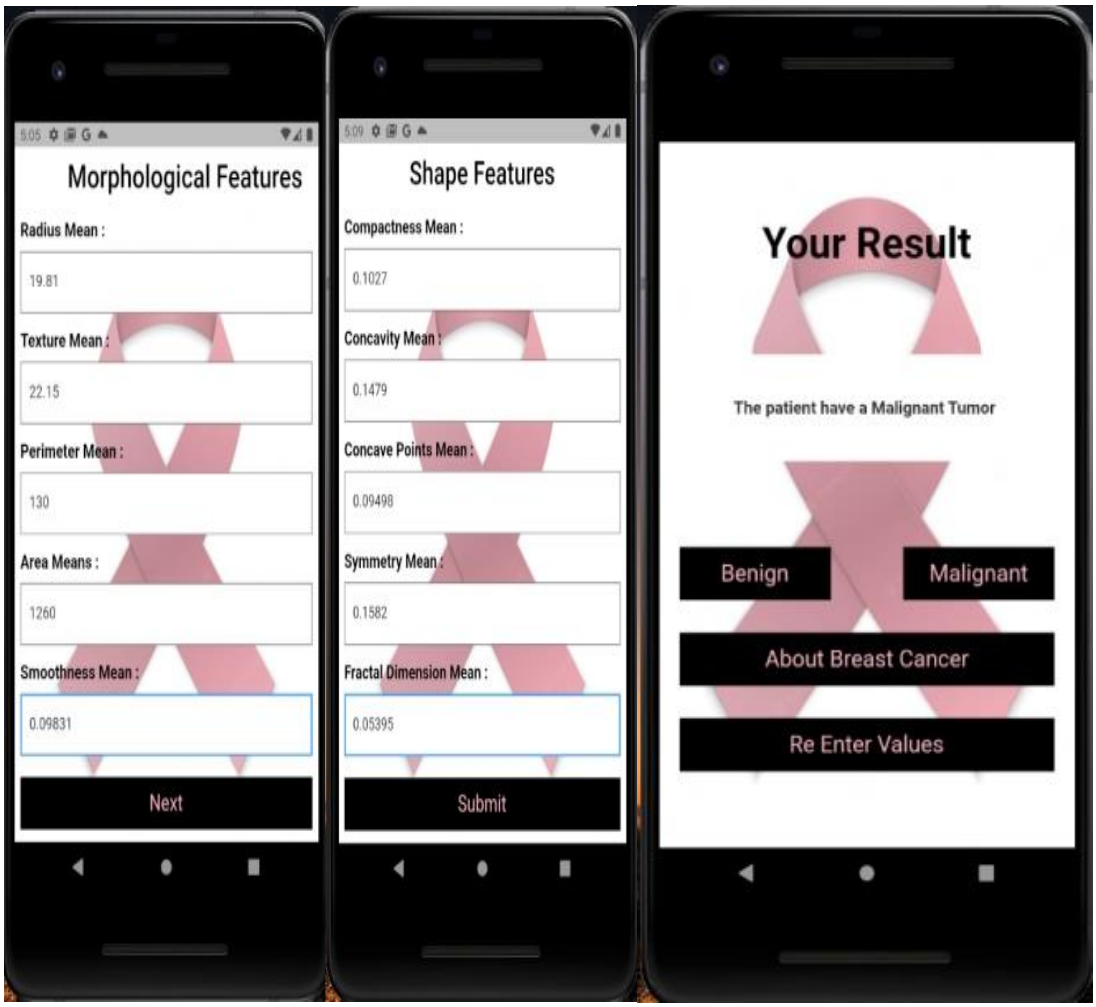
To test our app we will use Two samples from the data set, one of malignant samples and one of benign samples, we will enter this data into the application for testing.

20	849014	M	19.81	22.15	130	1260	0.09831	0.1027	0.1479	0.09498	0.1582	0.05395
21	8510426	B	13.54	14.36	87.46	566.3	0.09779	0.08129	0.06664	0.04781	0.1885	0.05766

Benign test: -



Malignant Test: -



References

- Mayo Clinic: -

https://www.mayoclinic.org/diseases-conditions/breast-cancer/symptoms-causes/syc-20352470

- CDC: -

https://www.cdc.gov/cancer/breast/basic_info/what-is-breast-cancer.htm

- Chat GPT: -

https://chat.openai.com

- Wikipedia: -

https://en.wikipedia.org/wiki/Flutter_(software)

https://en.wikipedia.org/wiki/Artificial_intelligence

- Kaggle: -

https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset

- Instaflutter: -

https://instaflutter.com/flutter-tutorials/create-flutter-app

- Flutter: -

https://docs.flutter.dev

- Postman: -

https://www.postman.com