औद्योगिक प्रशिक्षण के लिए राष्ट्रीय संस्थान

National Institute for Industrial Training

One Premier Organization with Non Profit Status | Registered Under Govt. of WB
Empanelled Under Planning Commission Govt. of India
Inspired By: National Task Force on IT & SD Government of India

National Institute for Industrial Training- One Premier Organization with Non Profit Status Registered Under Govt.of West Bengal,Empanelled Under Planning Commission Govt.of India , Empanelled Under Central Social Welfare Board Govt. of India , Registered with National Career Services , Registered with National Employment Services.

STROKE PREDICTION USING MACHINE LEARNING

Subject: Python with Machine Learning and Data Science

Submitted on: 07.03.2021

Submitted To: Soumotanu Mazumdar

Submitted by: Sreyashi Bandyopadhyay

STUDENT PROFILE

Name: SREYASHI BANDYOPADHYAY

College: **HERITAGE** INSTITUTE OF TECHNOLOGY

Course: Python with Machine Learning and Data Science

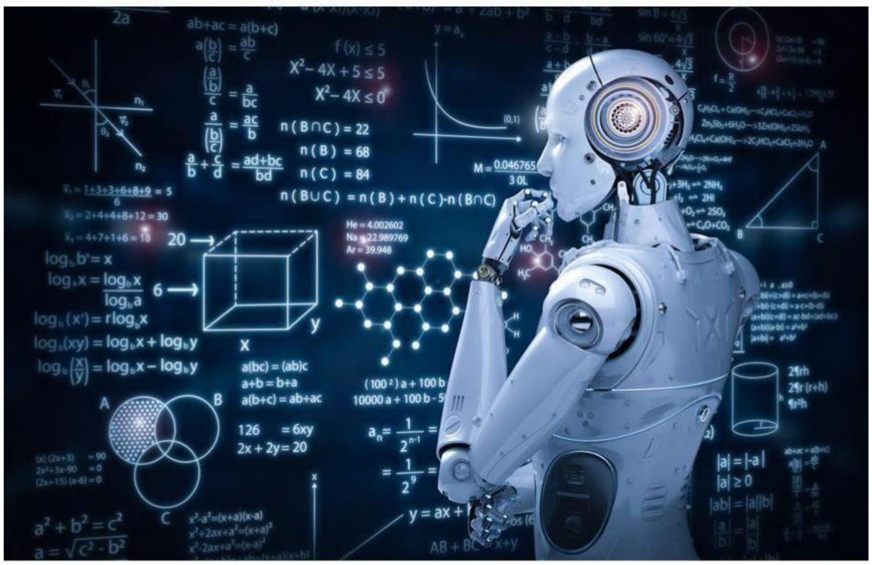
Year: 2^{nd} Year, 3^{rd} Semester

Year of Passing: 2023

E-mail ID: sreyashiban14@gmail.com

Ph Number: 9051529370


AI


ab+ac = a(b+c)
f (x) ≤ 5
X²– 4X + 5 ≤ 5
X²– 4X ≤ 0
n (B∩C) = 22
n (B) = 68
n (C) = 84
n (B∪C) = n (B) + n (C)-n (B∩C)
20
6
x
y
He = 4.002602
Ar = 39.948
a(bc) = (ab)c
a+b = b+a
a(b+c) = ab+ac
126 = 6xy
2x + 2y = 20
A
B
C
y = ax +
AB + BC

CONTENTS:

Acknowledgment

The success and final outcome of this project required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my project. All that I have done is only due to such supervision and assistance and I would not forget to thank them.

I respect and thank Mr. Avik Ghosh, for providing me an opportunity to do the project work and giving us all support and guidance which made me complete the project duly. I am extremely thankful to him for providing such a nice support and guidance, although he had busy schedule managing the corporate affairs.

I would like to express my heartfelt thanks and gratitude to my instructor Mr.Soumoutanu Mazumdar of National Institute for Industrial Training who gave me the golden opportunity to do this project and guided me in an exemplary manner. It helped me in doing a lot of Research and I came to know about a lot of things related to this topic.

Last but not the least, I thank my group members who shared necessary information and useful web links for preparing our project. It would not be possible to complete the project without the support of our friends , teachers and group members.

Objectives:

The main objectives of the project are as given below:

1. DATA MINING- Data mining is a process of discovering patterns in large data sets involving methods at the intersection of machine learning, statistics and database systems Data mining is an interdisciplinary subfield of computer science and statistics with an overall goal to extract information (with intelligent methods) from a data set and transform the information into a comprehensible structure for further use.

2. DATA CLEANING-Data cleaning is the process of preparing data for analysis by removing or modifying data that is incorrect, incomplete, irrelevant, duplicated, or improperly formatted.

3. DATA PREPROCESSING- Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues.

4. EXPLORATORY DATA ANALYSIS - In statistics, exploratory data analysis is an approach to analyzing data sets to summarize their main characteristics, often using statistical graphics and other data visualization methods.

5. DIVIDING INTO TRAINING AND TESTING SET- Data splitting is the act of partitioning available data into two portions; usually for cross-validatory purposes. One portion of the data is used to develop a predictive model and the other to evaluate the model's performance.

6. APPLYING VARIOUS CLASSIFICATION MODELS- Various classification models were used to predict the probability of stroke in patients , after taking into consideration various other factors.

7. EVALUATION OF RESULTS - The accuracy and precision of different models was evaluated using a confusion matrix and classification report.

INTRODUCTION

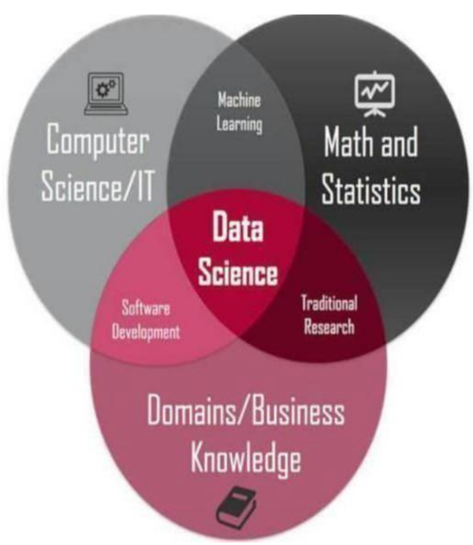
DATA SCIENCE:

Data science is the domain of study that deals with vast volumes of data using modern tools and techniques to find unseen patterns, derive meaningful information, and make business decisions. Data science uses complex machine learning algorithms to build predictive models.

The data used for analysis can be from multiple sources and present in various formats. Data science or data-driven science enables better decision making, predictive analysis, and pattern discovery.

Data science can add value to any business who can use their data well. From statistics and insights across workflows and hiring new candidates, to helping senior staff make better-informed decisions, data science is valuable to any company in any industry.

By extrapolating and sharing these insights, data scientists help organizations to solve vexing problems. Combining computer science, modeling, statistics, analytics, and math skills— along with sound business sense— data scientists uncover the answers to major questions that help organizations make objective decision



MACHINE LEARNING

Machine learning is a type of technology that aims to learn from experience. For example, as a human, you can learn how to play chess simply by observing other people playing chess. In the same way, computers are programmed by providing them with data from which they learn and are then able to predict future elements or conditions.

There are various steps involved in machine learning:

1. collection of data
2. filtering of data
3. analysis of data
4. algorithm training
5. testing of the algorithm
6. using the algorithm for future predictions

Machine learning uses different kinds of algorithms to find patterns, and these algorithms are classified into two groups:

- supervised learning
- unsupervised learning

Supervised Learning

Supervised learning is the science of training a computer to recognize elements by giving it sample data. The computer then learns from it and is able to predict future datasets based on the learned data.

For example, you can train a computer to filter out spam messages based on past information.

Supervised learning has been used in many applications, e.g. Facebook, to search images based on a certain description. You can now search images on Facebook with words that describe the contents of the photo. Since the social networking site already has a

database of captioned images, it is able to search and match the description to features from photos with some degree of accuracy.

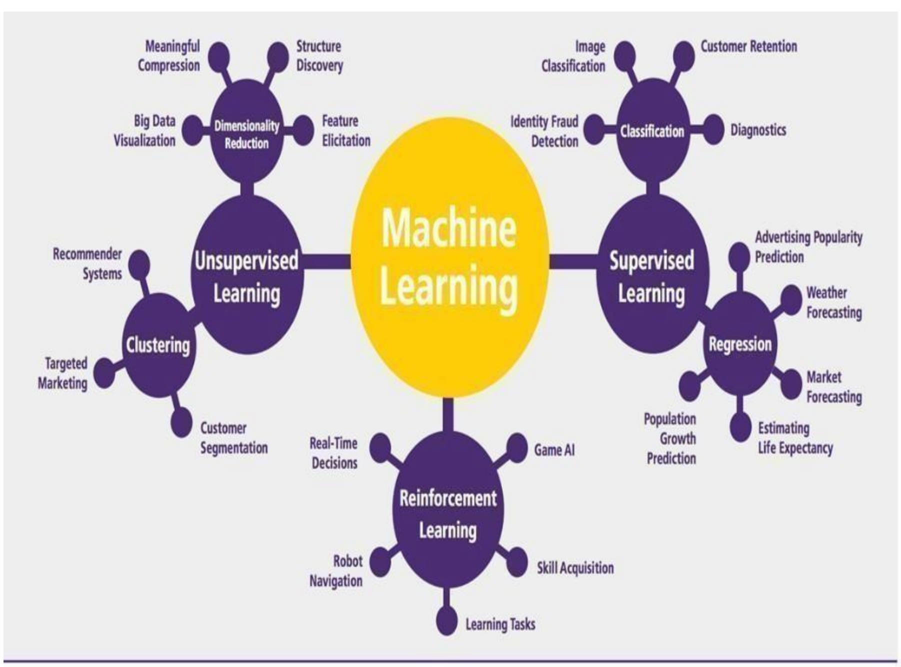
There are only two steps involved in supervised learning:

- training

- testing

Some of the supervised learning algorithms include:

- decision trees
- support vector machines
- naive Bayes
- k-nearest neighbor
- linear regression



PYTHON

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991.
Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library. Two major versions of Python are currently in active use:

Python 3.x is the current version and is under active development. Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python’s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

IMPORTS:

The libraries that have been imported for this project are as stated as follows.

1. Numpy- NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

2. Seaborn - Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

3. Pandas- In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

4. Matplotlib - Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+ .

5. TensorFlow -TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. Tensorflow is a symbolic math library based on dataflow and differentiable programming.

6. **Keras- Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.**

7. **Sklearn- The Sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.**

Advantages:

Advantages of Python

1. Easy Syntax
2. Readability
3. High-Level Language
4. Object-oriented programming
5. It's Open source and Free
6. Cross-platform
7. Widely Supported
8. It's Safe
9. Batteries Included
10. Extensible

Let's discuss about Advantages of Python in detail.

Easy Syntax of Python

Python's syntax is easy to learn, so both non-programmers and programmers can startprogramming right away.

Very Clear Readability of Python

Python's syntax is very clear, so it is easy to understand program code. (Python is often referred to as "executable pseudo-code" because its syntax mostly follows the conventions used by programmers tooutline their ideas without the formal verbosity of code in most programming languages.
In other words, syntax of Python is almostidentical to the simplified "pseudo-code" used by many programmers to prototype and describe theirsolution to other programmers. Thus Python can be used to prototype and test code which is later to be implemented in other programming languages).

Python High-Level Language

Python looks more like a readable, human language than like a low-level language. This gives you the ability to program at a faster rate than a low-level language will allow you.

Python Is Open-Source and Free

Python is both free and open-source. The Python Software Foundation distributes pre -made binaries that are freely available for use on all major operating systems called CPython. You can get CPython's source-code, too. Plus, we can modify the source code and distribute as allowed by CPython's license.

Python is a Cross-platform

Python runs on all major operating systems like Microsoft Windows, Linux, and Mac OS X.

Python Object-oriented programming

Object-oriented programming allows you to create data structures that can be reused, which reduces the amount of repetitive work that you'll need to do. Programming languages usually define objects with namespaces, like class or def, and objects can edit themselves by using keyword, like this or self.
Most modern programming languages are object-oriented (such as Java, C++, and C#) or have support for OOP features (such as Perl version 5 and later).
Additionally, object-oriented techniques can be used in the design of almost any non-trivial software and implemented in almost any programming or scripting language.
Python's support for object-oriented programming is one of its greatest benefits to new programmers because they will be encountering the same concepts and terminology in their work environment. If you ever decide to switch languages or use any other for that fact, you'll have a significant chance that you'll be working with object-oriented programming.

Python Widely Supported Programming Language

Python has an active support community with many websites, mailing lists, and USENET “netnews” groups that attract a large number of knowledgeable and helpful contributors.

Python is a Safe

Python doesn’t have pointers like other C-based languages, making it much more reliable. Along with that, errors never pass silently unless they’re explicitly silenced. This allows you to see and read why the program crashed and where to correct your error.

Python Batteries Included Language

Python is famous for being the “batteries are included” language. There are over 300 standard library modules which contain modules and classes for a wide variety of programming tasks.
For example, the standard library contains modules for safely creating temporary files (named or anonymous), mapping files into memory (including use of shared and anonymous memory mappings), spawning and controlling sub-processes, compressing and decompressing files (compatible with gzip or PK-zip) and archives files (such as Unix/Linux “tar”).
Accessing indexed “DBM” (database) files, interfacing to various graphical user interfaces (such as the TK toolkit and the popular WxWindows multi-platform windowing system), parsing and maintaining CSV (comma-separated values) and “.cfg” or “.ini” configuration files (similar in syntax to the venerable WIN.INI files from MS-DOS and MS-Windows), for sending e-mail, fetching and parsing web pages, etc. It’s possible, for example, to create a custom web server in Python using less than a dozen lines of code, and one of the standard libraries, of course.

Python is Extensible

In addition to the standard libraries there are extensive collections of freely available add-on modules, libraries, frameworks, and tool-kits. These generally conform to similar standards and conventions. For example, almost all of the database adapters (to talk to almost any client-server RDBMS engine such as MySQL, Postgres, Oracle, etc) conform to the Python DBAPI and thus can mostly be accessed using the same code. So it’s usually easy to modify a Python program to support any database engine.

Future Scopes:

Python is one of the fastest growing languages and has undergone a successful span of more than 25 years as far as its adoption is concerned. This success also reveals a promising future scope of python programming language.

In fact, it has been continuously serving as the best programming language for application development, web development, game development, system administration, scientific and numeric computing, GIS and Mapping etc.

Popularity of python

The reason behind the immense popularity of python programming language across the globe is the features it provides. Have a look at the features of python language.

(1) Python Supports Multiple Programming Paradigms

Python is a multi-paradigm programming language including features such as object-oriented, imperative, procedural, functional, reflective etc.

(2) Python Has Large Set Of Library and Tools

Python has very extensive standard libraries and tools that enhance the overall functionality of python language and also helps python programmers to easily write codes. Some of the important python libraries and tools are listed below.

- Built-in functions, constants, types, and exceptions.
- File formats, file and directory access, multimedia services.
- GUI development tools such as Tkinter
- Custom Python Interpreters, Internet protocols and support, data compression and archiving, modules etc.
- Scrappy, wxPython, SciPy, matplotlib, Pygame, PyQT, PyGTK etc.

(3) Python Has a Vast Community Support

This is what makes python a favorable choice for development purposes. If you are having problems writing python a program, you can post directly to python community and will get the response with the solution of your problem. You will also find many new ideas regarding python technology and change in the versions.

(4) Python is Designed For Better Code Readability

Python provides a much better code readability as compared to another programming language. For example, it uses whitespace indentation in place of curly brackets for delimiting the block of codes. Isn’t it awesome?

(5) Python Contains Fewer Lines Of Codes
Codes are written in python programming language complete in fewer lines thus reducing the efforts of programmers. Let's have a look on the following "Hello World" program written in C, C++, Java, and Python.
While, C, C++, and Java take six, seven and five lines respectively for a simple "Hello World" program. Python takes only a single line which means, less coding effort and time is required for writing the same program.

Future Technologies Counting On Python

Generally, we have seen that python programming language is extensively used for web development, application development, system administration, developing games etc.
But do you know there are some future technologies that are relying on python? As a matter of fact, Python has become the core language as far as the success of these technologies is concerned. Let's dive into the technologies which use python as a core element for research, production and further developments.
(1) Artificial Intelligence (AI)

Python programming language is undoubtedly dominating the other languages when future technologies like Artificial Intelligence (AI) come into the play.
There are plenty of python frameworks, libraries, and tools that are specifically developed to direct Artificial Intelligence to reduce human efforts with increased accuracy and efficiency for various development purposes.
It is only the Artificial Intelligence that has made it possible to develop speech recognition system, autonomous cars, interpreting data like images, videos etc.

We have shown below some of the python libraries and tools used in various Artificial Intelligence branches.

- Machine Learning- PyML, PyBrain, scikit-learn, MDP Toolkit, GraphLab Create, MIPy etc.
- General AI- pyDatalog, AIMA, EasyAI, SimpleAI etc.
- Neural Networks- PyAnn, pyrenn, ffnet, neurolab etc.
- Natural Language & Text Processing- Quepy, NLTK, gensim

(2) Big Data

The future scope of python programming language can also be predicted by the way it has helped big data technology to grow. Python has been successfully contributing in analyzing a large number of data sets across computer clusters through its high-performance toolkits and libraries.

Let's have a look at the python libraries and toolkits used for Data analysis and handling other big data issues.

- Pandas
- Scikit-Learn
- NumPy
- SciPy
- GraphLab Create
- IPython
- Bokeh
- Agate
- PySpark
- Dask

(2) Networking

Networking is another field in which python has a brighter scope in the future. Python programming language is used to read, write and configure routers and switches and perform other networking automation tasks in a cost-effective and secure manner.

For these purposes, there are many libraries and tools that are built on the top of the python language. Here we have listed some of these python libraries and tools especially used by network engineers for network automation.

- Ansible
- Netmiko
- NAPALM(Network Automation and Programmability Abstraction Layer with Multivendor Support)
- Pyeapi
- Junos PyEZ
- PySNMP
- Paramiko SSH

Real-Life Python Success Stories

Python has seemingly contributed as a core language for increasing productivity regarding various development purposes at many of the IT organizations. We have shown below some of the real-life python success stories.

- Australia's RMA Department D-Link has successfully implemented python for creating DSL Firmware Recovery System.
- Python has helped Gusto.com, an online travel site, in reducing development costs and time.
- ForecastWatch.com also uses python in rating the accuracy of weather forecast reports provided by companies such as Accuweather, MyForecast.com and The Weather Channel.
- Python has also benefited many product development companies such as Acqutek, AstraZeneca, GravityZoo, Carmanah Technologies Inc. etc in creating autonomous devices and software.
- Test&Go uses python scripts for Data Validation.
- Industrial Light & Magic(ILM) also uses python for batch processing that includes modeling, rendering and compositing thousands of picture frames per day.

There is a huge list of success stories of many organizations across the globe which are using python for various purposes such as software development, data mining, unit testing, product development, web development, data validation, data visualization etc.

These success stories directly point towards a promising future scope of python programming language.

Top Competitors Of Python

The future scope of python programming language also depends on its competitors in the IT market. But, due to the fact that it has become a core language for future technologies such as artificial intelligence, big data, etc., it will surely rise further and will be able to beat its competitors.

Hardware and Software Requirements :

Software Requirements:

Operating System: Windows/Linux
Front End: Python 3.7
Platform: Anaconda

Hardware requirements:

Speed: 233MHz and above
Hard disk: 10GB
RAM: 256 MB

<\SOURCE CODE>

Min Max Scaling:

The first intuitive option is to use what is called the Min-Max scaler. In this we subtract the Minimum from all values – thereby marking a scale from Min to Max. Then divide it by the difference between Min and Max. The result is that our values will go from zero to 1. This is quite acceptable in cases where we are not concerned about the standardization along the variance axes. e.g. image processing or neural networks expecting values between 0 to 1.The downside however is that because we have now bounded the range from 0 to 1, we will have lower standard deviations and it suppresses the effect of outliers.

Standard Scaler:

The way to overcome this is through Standard Scaler – or z-score normalization. Firstly by subtracting the mean it brings the values around 0 – so has zero mean. Secondly, it divides the values by standard deviation thereby ensuring that the resulting distribution is standard with a mean of 0 and standard deviation of

So where would you use Standard Scaler against Min Max:

The answer is as always 'it depends' but here are some general guidelines:

For most cases Standard Scaler would do no harm. Especially when dealing with variance (PCA, clustering, logistic regression, SVMs, perceptrons, neural networks) in fact Standard Scaler would be very important. On the other hand it will not make much of a difference if you are using tree based classifiers or regressors. My bias is to default to Standard Scaling and check if I need to change it.

Normalization

Feature Scaling is an essential step in the data analysis and preparation of data for modeling. Wherein, we make the data scale-free for easy analysis.

Normalization is one of the feature scaling techniques. We particularly apply normalization when the data is skewed on the either axis i.e. when the data does not follow the Gaussian distribution.

In normalization, we convert the data features of different scales to a common scale which further makes it easy for the data to be processed for modeling. Thus, all the data features(variables) tend to have a similar impact on the modeling portion.

STANDARDISATION

Standardization is used on the data values that are normally distributed. Further, by applying standardization, we tend to make the mean of the dataset as 0 and the standard deviation equivalent to 1.

That is, by standardizing the values, we get the following statistics of the data distribution

mean = 0

standard deviation = 1

Let us now focus on the various ways of implementing Standardization:

1. Using preprocessing. scale() function

The preprocessing.scale(data) function can be used to standardize the data values to a value having mean equivalent to zero and standard deviation as 1.

Here, we have loaded the IRIS dataset into the environment using the below line:

2. Using StandardScaler() function

Python sklearn library offers us with StandardScaler() function to perform standardization on the dataset. Here, again we have made use of Iris dataset.

Further, we have created an object of StandardScaler() and then applied fit_transform() function to apply standardization on the dataset.

In [1]:
```python
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```python
df=pd.read_csv('stroke_data.csv')
```

In [3]:
```python
df.head()
```

Out[3]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_lev
0	Male	58.0	1	0	Yes	Private	Urban	87.9
1	Female	70.0	0	0	Yes	Private	Rural	69.0
2	Female	52.0	0	0	Yes	Private	Urban	77.5
3	Female	75.0	0	1	Yes	Self-employed	Rural	243.5
4	Female	32.0	0	0	Yes	Private	Rural	77.6

This is our given data and the task is to predict the probability of stroke in a person , given their lifestyle , medical history, age , genderetc

EXPLORATORY DATA ANALYSIS - The following section is aimed at exploring the given data and finding out meaningful relationships among the various parameters to predict stroke probability in a person. This also represents how the probability varies according to age , gender , lifestyle choice s, job type , medical conditions etc through the use of plotting tools of seaborn library like graphs and charts to convey the conclusions better and in a visually appealing manner

Before we proceed to EDA , data cleaning has been done to examine if any null values are present in any column. If null values are present , the empty elements are fil led with 0 or NA to reduce discrepancies in EDA.

In [4]:
```python
df.isnull().sum()             #this gives sum of null value for each column
```

Out[4]:
```
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                  0
smoking_status       0
stroke               0
dtype: int64
```

Clearly , there are no null values in the csv file so we can readily perform EDA now

Exploring the age column :-

As we can see, the age column has extremely diverse values and hence it would make sense to make a new column in the file which groups people according to their age group. People having age 20 or 24 are likely to have the same conditions , so they are grouped together in a new column called "AGE GROUP"

In [5]:
```
df['AGE GROUP']=df['age'] //10
```

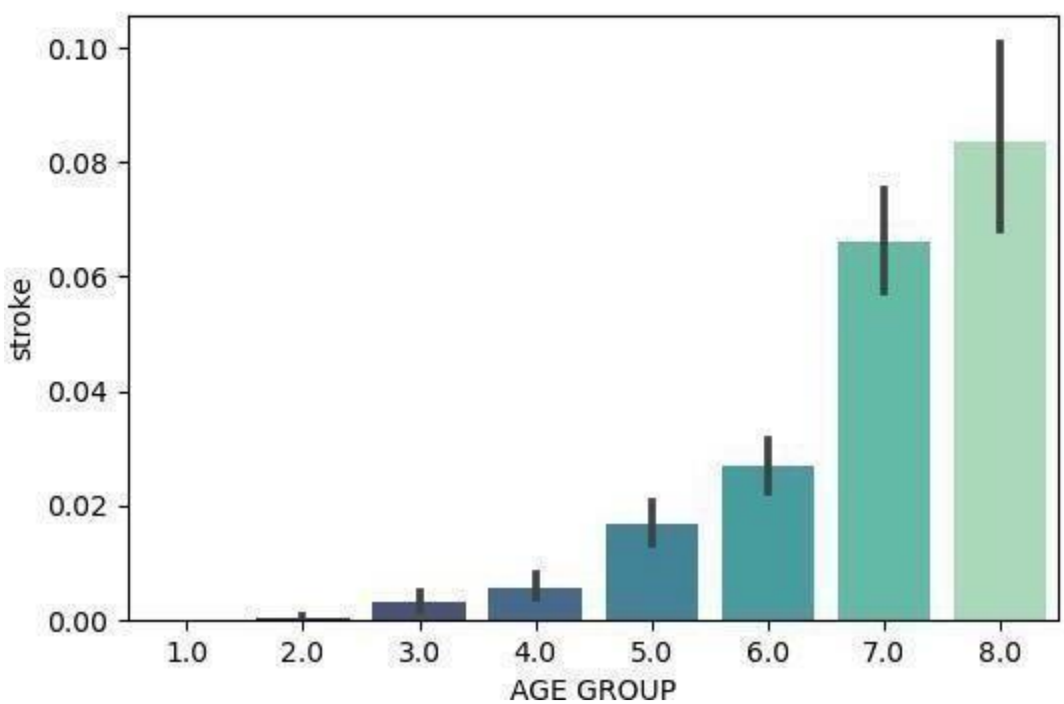
In [6]:
```
df.head()           #shows updated table with the new column that was created by the nam
```

Out[6]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_lev
0	Male	58.0	1	0	Yes	Private	Urban	87.9
1	Female	70.0	0	0	Yes	Private	Rural	69.0
2	Female	52.0	0	0	Yes	Private	Urban	77.5
3	Female	75.0	0	1	Yes	Self-employed	Rural	243.5
4	Female	32.0	0	0	Yes	Private	Rural	77.6

In [7]:
```
plt.figure(dpi=100)
sns.barplot(x='AGE GROUP' , y='stroke' ,data=df ,palette= 'mako')
```

Out[7]: <AxesSubplot:xlabel='AGE GROUP', ylabel='stroke'>

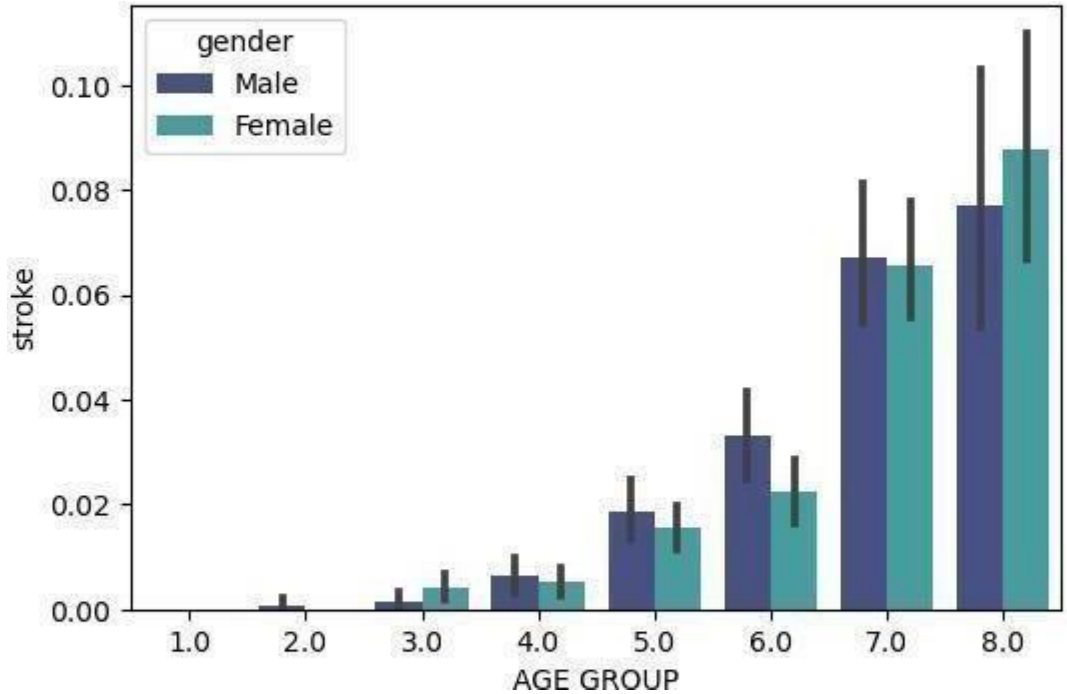


This is a bar graph with "age group" column as x axis and "stroke" as y axis. The bar graph shows that the greatest probability for stroke occurs in the age group of 80-89 years and this possibility gradually increases with a persons age.

Exploring the relationship between age and stroke probability across the various genders

In [8]:
```
plt.figure(dpi=100)
sns.barplot(x='AGE GROUP' , y='stroke' ,data=df ,palette= 'mako' , hue='gender')
```

Out[8]: <AxesSubplot:xlabel='AGE GROUP', ylabel='stroke'>

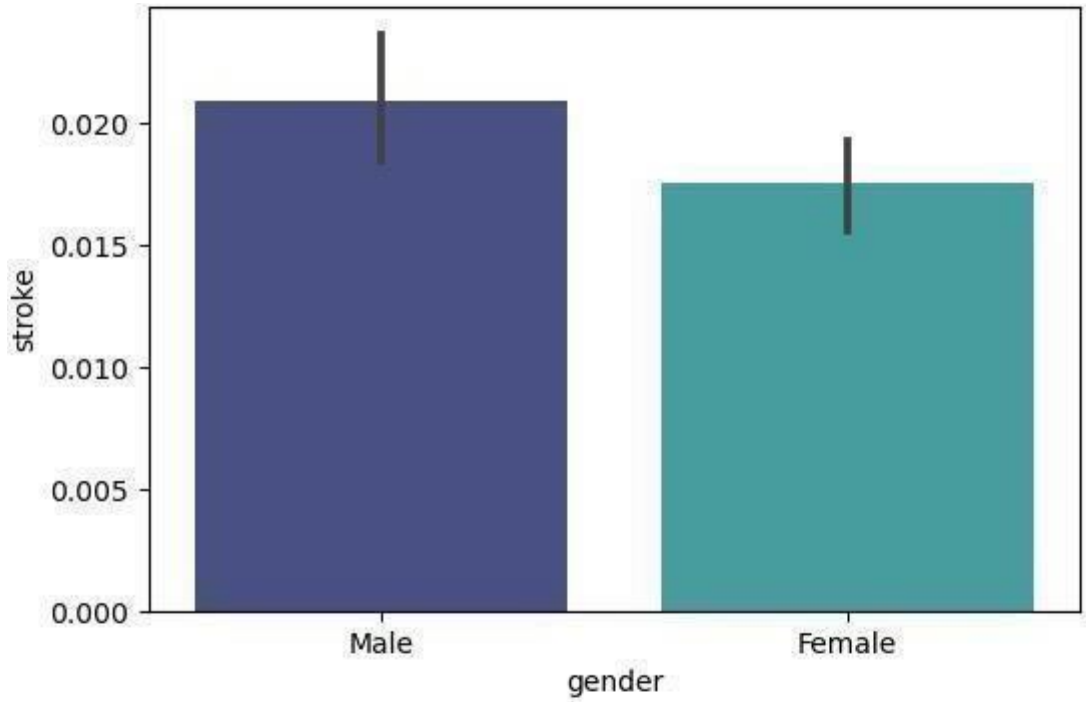


In almost all age groups , men are at greater risk than women , except for the age group 80 -89 years where women are at a significantly greater risk than men.

Exploring the relationship between the gender and the stroke probability

In [9]:
```
plt.figure(dpi=100)
sns.barplot(x='gender' , y='stroke' ,data=df ,palette= 'mako')
```

Out[9]: <AxesSubplot:xlabel='gender', ylabel='stroke'>

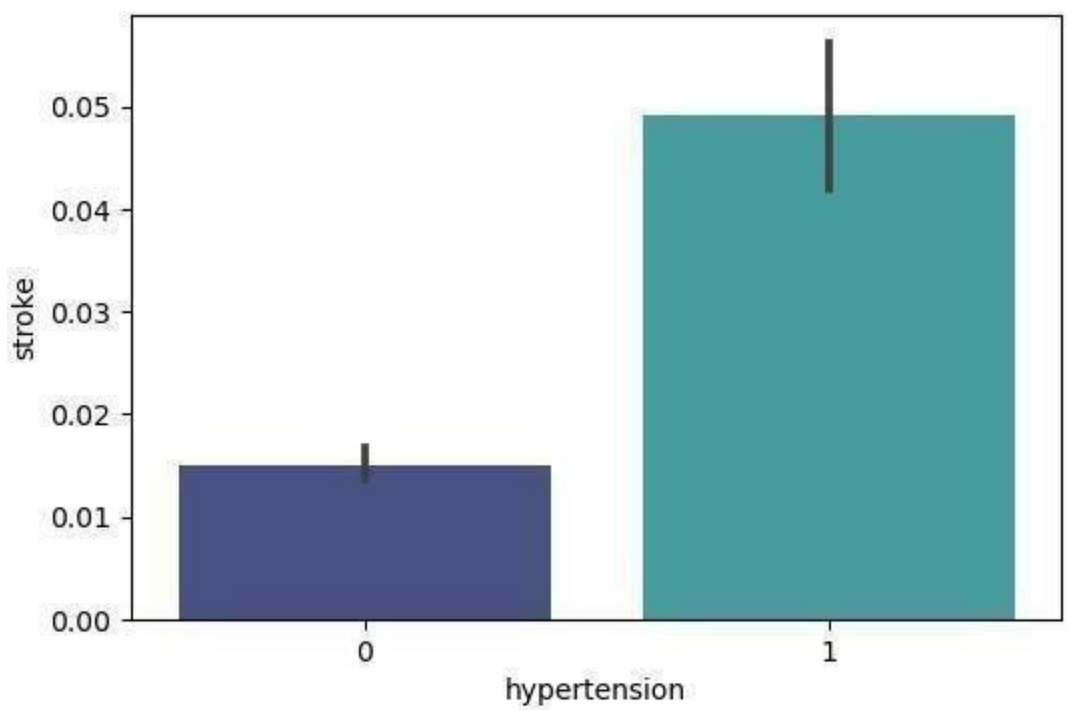


Clearly,malepatientsaremorelikelytosufferfromastrokethanfemalepatients.Probabilityfor malestosufferastrokeisabove0.02whiletheprobabilityforfemales isaround0.017

Exploring the relation between hypertension and stroke possibility

In [10]:
```
plt.figure(dpi=100)
sns.barplot(x="hypertension" , y="stroke" ,data=df ,palette= "mako")
```

Out[10]: <AxesSubplot:xlabel='hypertension', ylabel='stroke'>

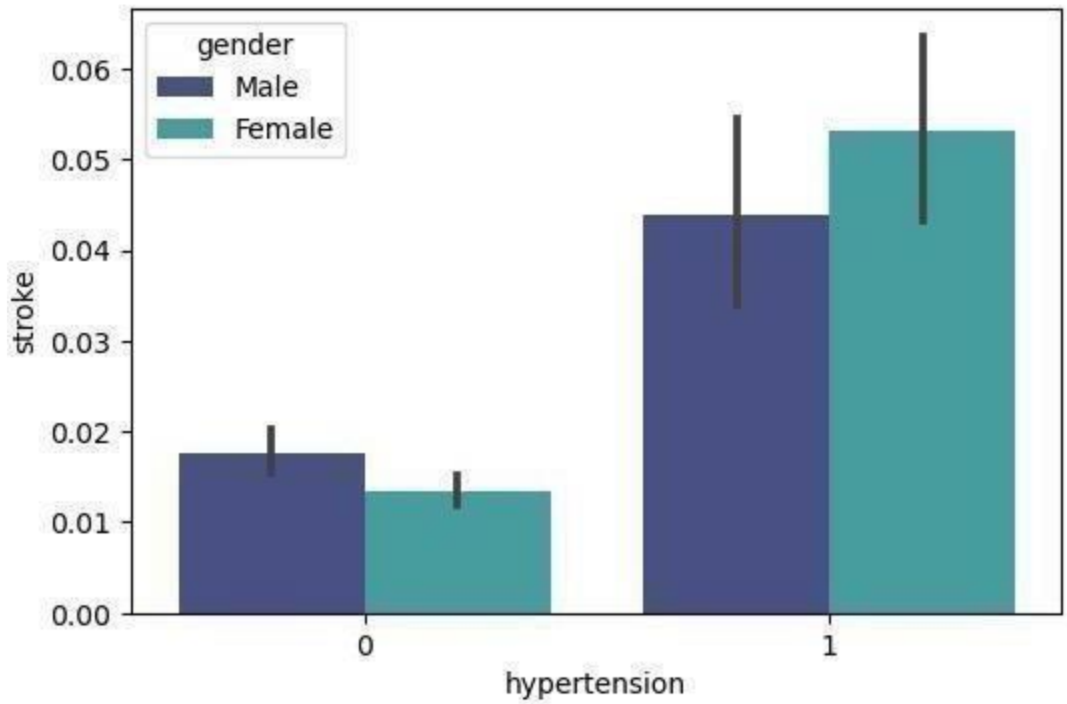


The conclusion drawn from the above bar graph is that people having hypertension are more probable to suffer a stroke than people who do not suffer from it. Here 0 indicates hypertension not present and 1 indicates hypertension present

Exploring the relation between hypertension and stroke possibility on the basis of gender of patient

In [11]:
```
plt.figure(dpi=100)
sns.barplot(x='hypertension' , y='stroke' ,data=df ,palette= 'mako' , hue='gender')
```

Out[11]: <AxesSubplot:xlabel='hypertension', ylabel='stroke'>

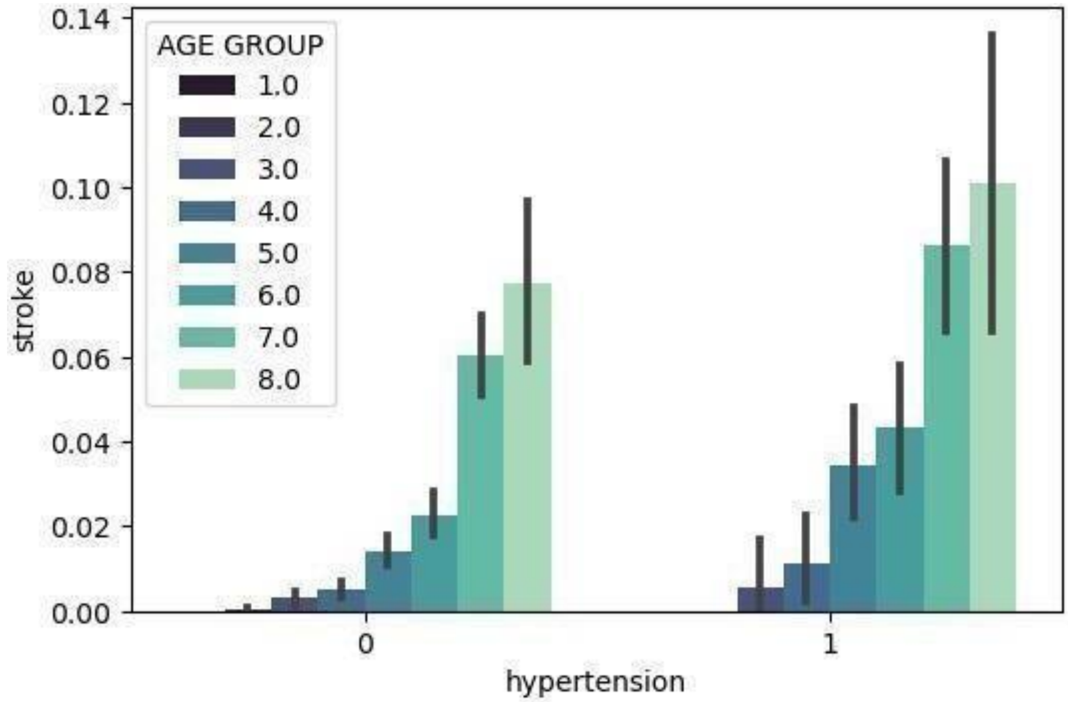


Female patients of hypertension more probable to suffer a stroke than male patients. Probability for female hypertension patients is above 0.05 whereas for males it is around 0.04. In case hypertension is absent , men are more likely to suffer from a stroke than females.

Exploring the relation between hypertension and stroke possibility on the basis of age group of patient

In [12]:
```
plt.figure(dpi=100)
sns.barplot(x='hypertension' , y='stroke' ,data=df ,palette= 'mako' , hue ='AGE GROUP')
```

Out[12]: <AxesSubplot:xlabel='hypertension', ylabel='stroke'>

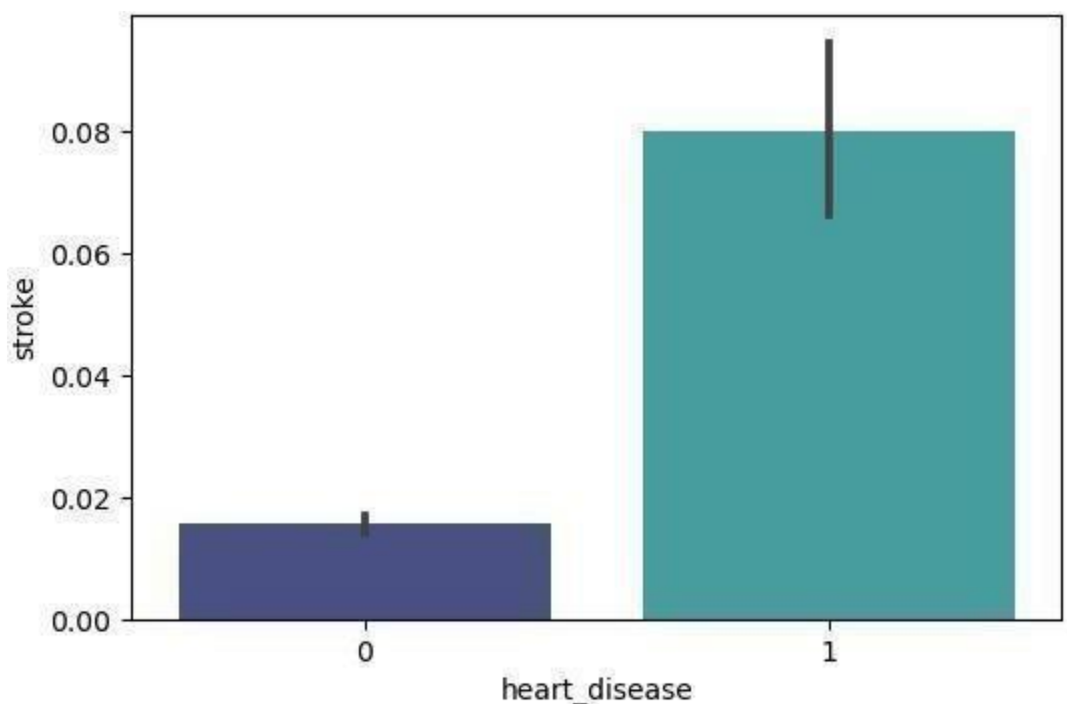


People who do not have hypertension , aged 80-89 years are more at risk from a stroke than patients in their 50's or 60's but suffering from hypertension

Exploring the relation between heart disease and stroke possibility

In [13]:
```
plt.figure(dpi=100)
sns.barplot(x='heart_disease' , y='stroke' ,data=df ,palette= 'mako')
```

Out[13]: <AxesSubplot:xlabel='heart_disease', ylabel='stroke'>



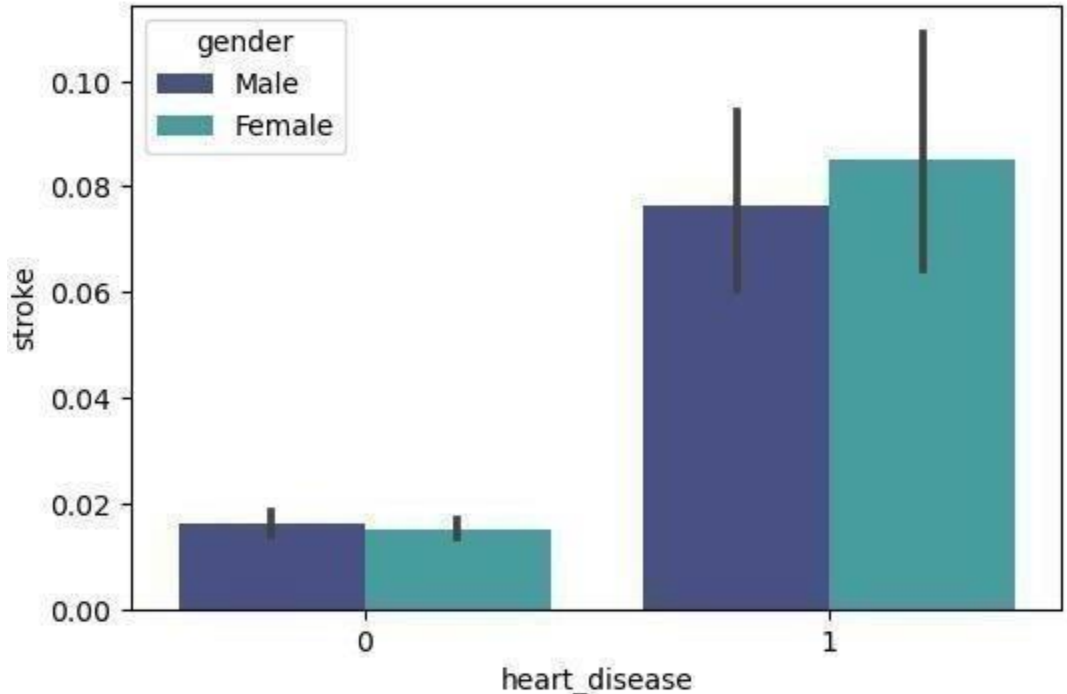
People having heart disease are much more likely to suffer a stroke than people not having heart disease. Probability of a person having heart disease to suffer froma stroke is around 0.08 whereas

the fraction is below 0.02 otherwise . This indicates a strong correlation between heart disease and stroke probability.

Exploring the relation between heart disease and stroke possibility on the basis of gender

In [14]:
```
plt.figure(dpi=100)
sns.barplot(x="heart_disease" , y="stroke" ,data=df ,palette= "mako" , hue="gender")
```

Out[14]: <AxesSubplot:xlabel='heart_disease', ylabel='stroke'>



The above graph indicates that females having a heart disease are more likely to suffer from a stroke than men having the heart disease.

Exploring the relation between the combination of heart disease and hypertension on the stroke probability

Creating a new column combined with data from column hypertension and heartdisease and putting 1 where a patient has both , 0 otherwise.

In [15]:
```
## finding intersection of two col heart disease and hypertension
df["combined"] =df["hypertension"] * df["heart_disease"]
```

In [16]:
```
df["combined"].unique()       #seeing all possible values of column "combined"
```

Out[16]: array([0, 1], dtype=int64)

In [17]:
```
df.head()       #new column "combined" has been added
```

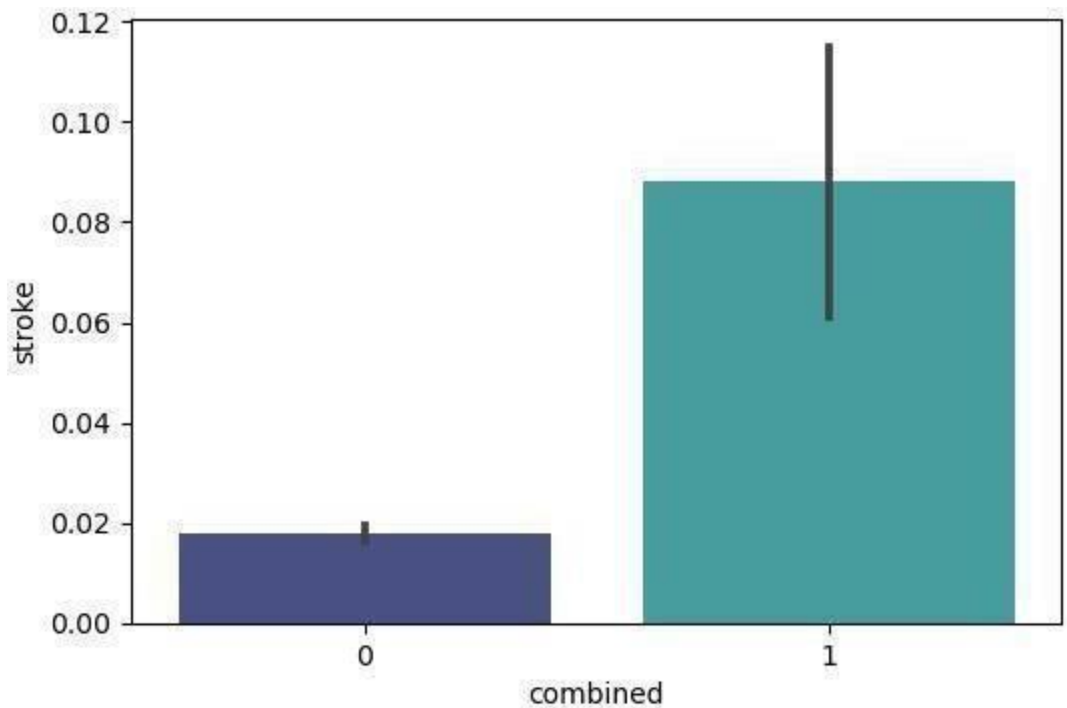
Out[17]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_lev
0	Male	58.0	1	0	Yes	Private	Urban	87.9
1	Female	70.0	0	0	Yes	Private	Rural	69.0

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_lev
2	Female	52.0	0	0	Yes	Private	Urban	77.5
3	Female	75.0	0	1	Yes	Self-employed	Rural	243.5
4	Female	32.0	0	0	Yes	Private	Rural	77.6

In [18]:
```
plt.figure(dpi=100)
sns.barplot(x='combined' , y='stroke' ,data=df ,palette= 'mako')
```

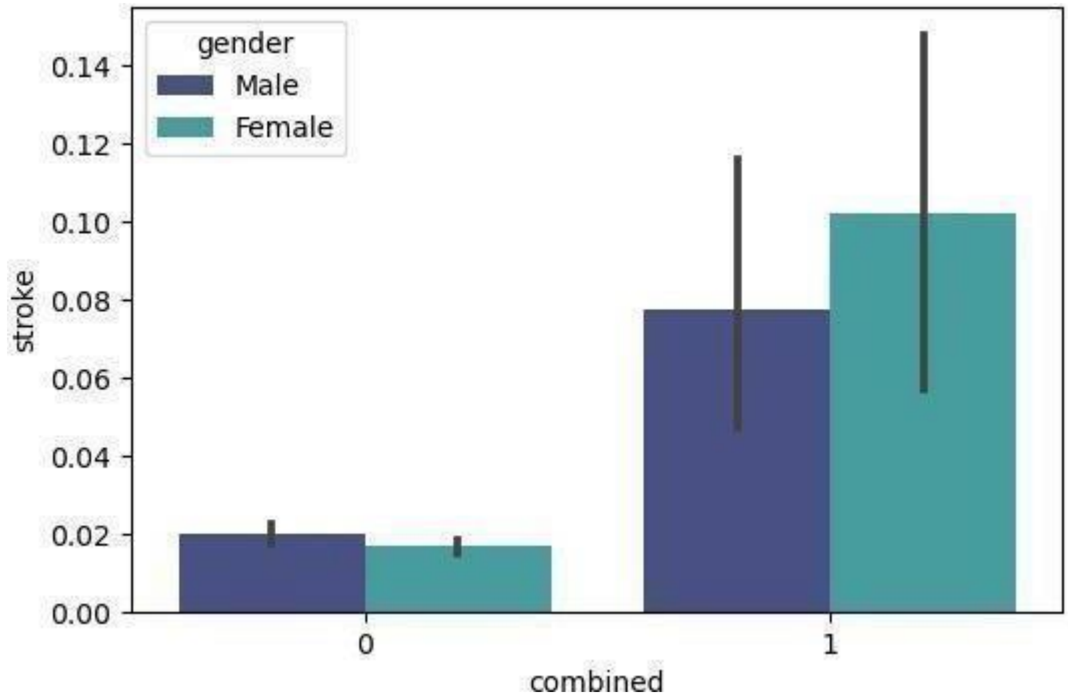
Out[18]:
```
<AxesSubplot:xlabel='combined', ylabel='stroke'>
```



Thus it shows that people having both heart disease and hypertension are much more likely to suffer a stroke (0.08) than people who do not have these conditions or have only one of them.

In [19]:
```
plt.figure(dpi=100)
sns.barplot(x='combined' , y='stroke' ,data=df ,palette= 'mako' , hue='gender')
```

Out[19]:
```
<AxesSubplot:xlabel='combined', ylabel='stroke'>
```

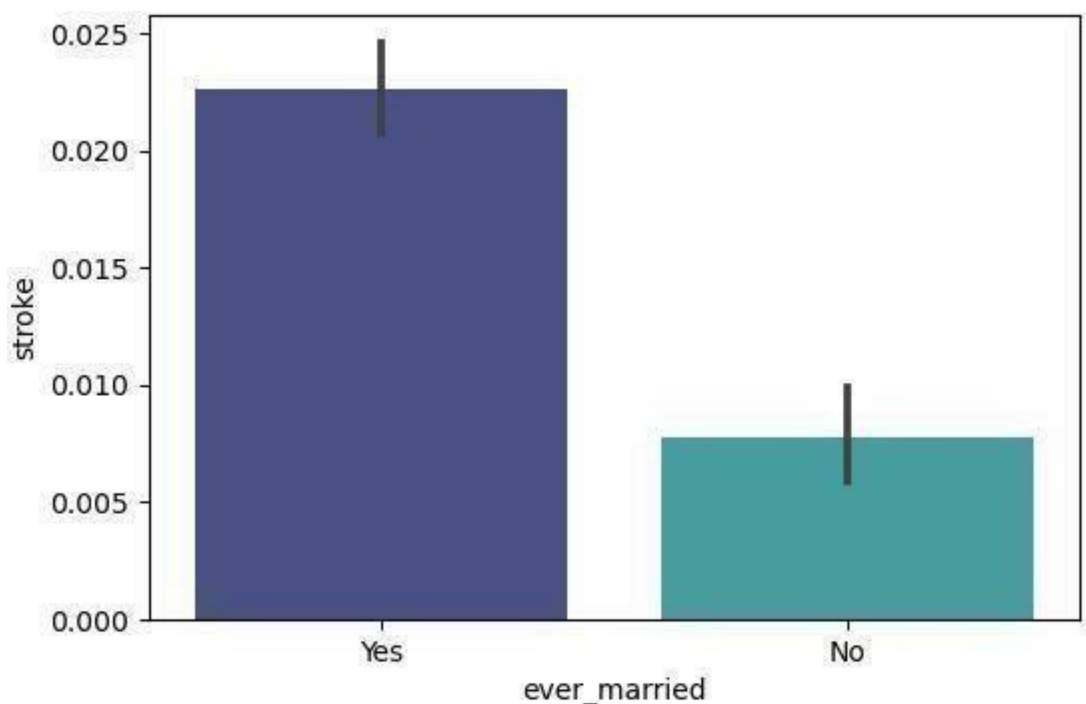


The above bar graph indicates that if both heart disease and hypertension is present , then females are at higher risk(0.11) than males(0.08 approx). In cases where only one disease is present or none is present , men are at greater risk ofstroke.

Exploring the relationship between marital status and stroke probability

In [20]:
```
plt.figure(dpi=100)
sns.barplot(x=df['ever_married'], y= df['stroke'] , palette = 'mako')
```

Out[20]: <AxesSubplot:xlabel='ever_married', ylabel='stroke'>

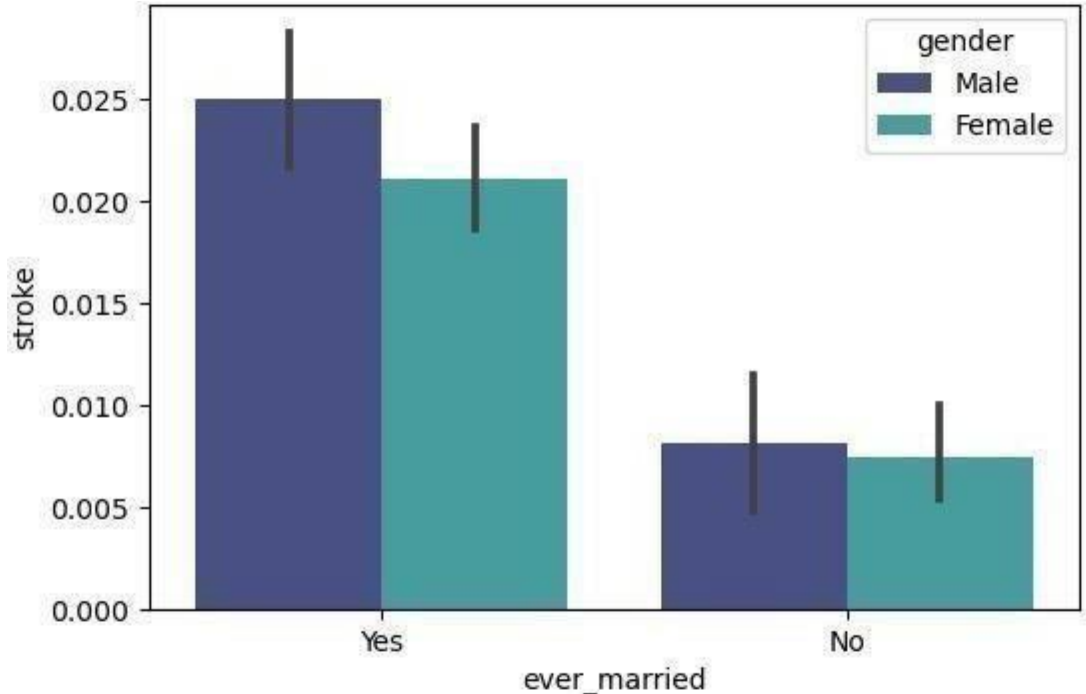


The above graph indicates that married people are more probable(0.023) to suffer from a stroke than unmarried people where stroke probability is around (0.06) .

Exploring the relationship between marital status and stroke probability , gender-wise

In [21]:
```
plt.figure(dpi=100)
sns.barplot(x="ever_married" , y="stroke" ,data=df ,palette= "mako" , hue="gender")
```

Out[21]: <AxesSubplot:xlabel='ever_married', ylabel='stroke'>

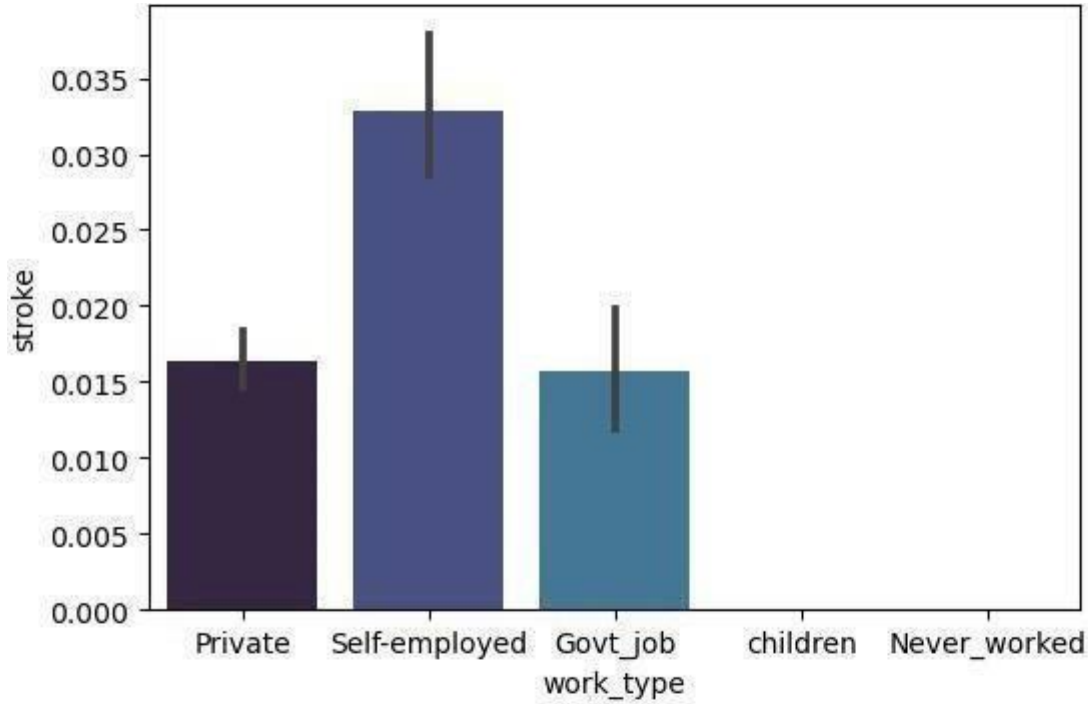


The conclusions drawn from this graph are as follows. Men are more likely to suffer a stroke than women , whether married or unmarried. Unmarried people significantly less likely to suffer from a stroke.

Exploring relationship between work type and stroke

In [22]:
```
plt.figure(dpi=100)
sns.barplot(x="work_type" , y="stroke" ,data=df ,palette= "mako")
```

Out[22]: <AxesSubplot:xlabel='work_type', ylabel='stroke'>



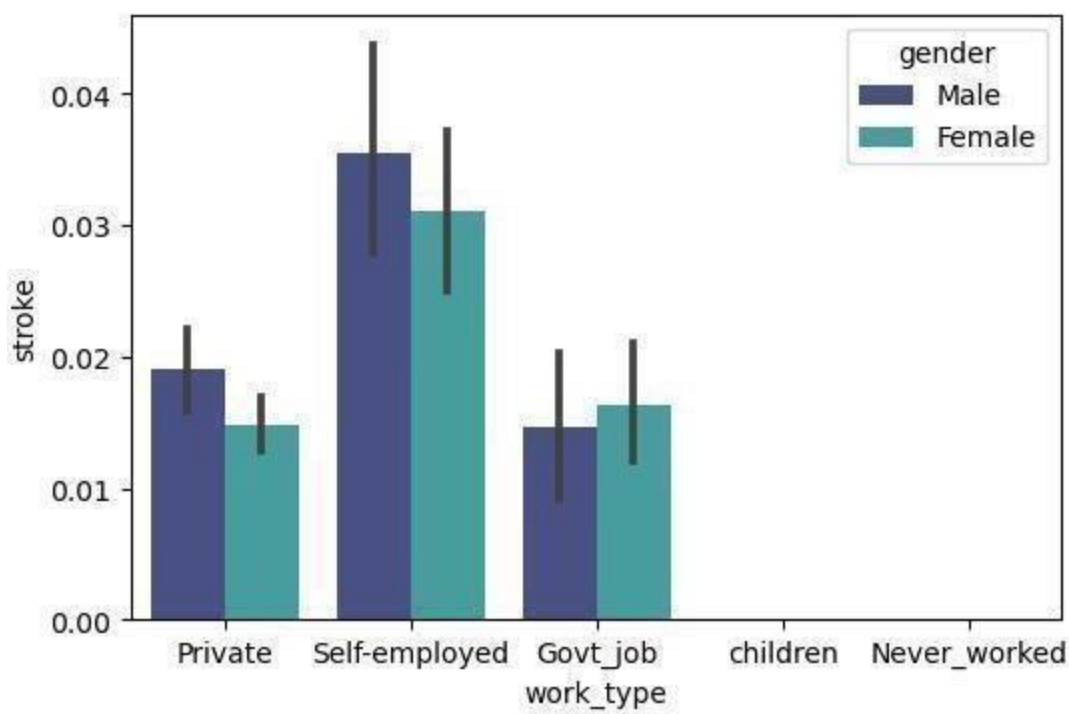
The bar graph shows that self employed people have the greatest probability of a stroke (0.033), followed by people working in the private sector(probability is 0.016 approx). People having government jobs least likely to suffer from a stroke among all the peole who are working.

Probabilityofchildrenandpeoplewhohaveneverbeenemployed isvery less(close to 0.0)

Exploring relationship between work type and stroke on the basis of gender

In [23]:
```
plt.figure(dpi=100)
sns.barplot(x='work_type' , y='stroke' ,data=df ,palette= 'mako' , hue='gender')
```

Out[23]: <AxesSubplot:xlabel='work_type', ylabel='stroke'>



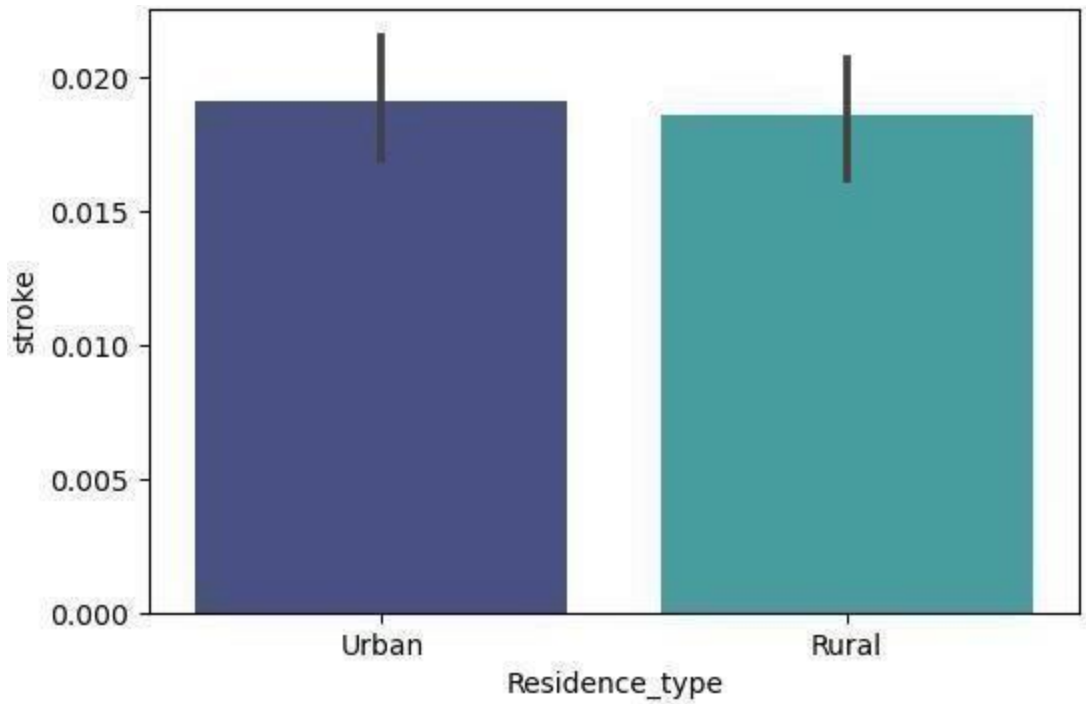
Amonggenderandworktypemenmorelikelytosufferfromstrokeineveryworktypeotherthan

govt job where females are at greater risk

Exploring relationship between residence type and stroke probability

In [24]:
```
plt.figure(dpi=100)
sns.barplot(x="Residence_type" , y="stroke" ,data=df ,palette= "mako" )
```

Out[24]: <AxesSubplot:xlabel='Residence_type', ylabel='stroke'>

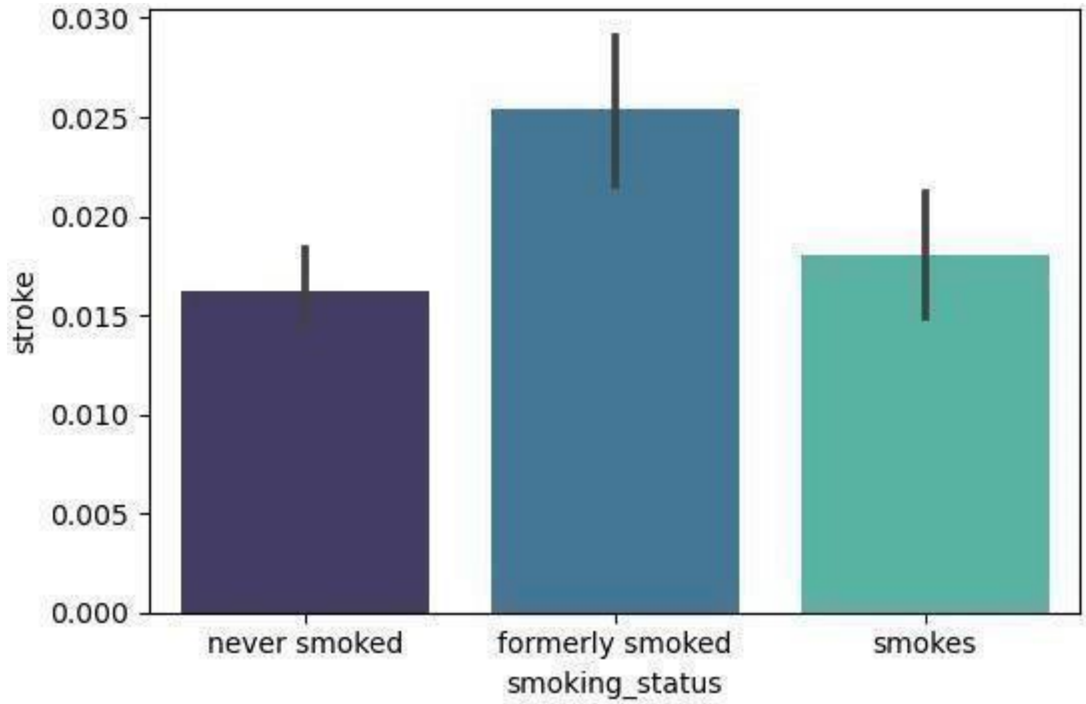


It can be clearly seen that people living in rural areas are at slightly lower risk than urban city-dwellers.

Exploring relationship between stroke and smoking status of patients

In [25]:
```
plt.figure(dpi=100)
sns.barplot(x="smoking_status" ,y="stroke" ,data=df ,palette="mako")
```

Out[25]: <AxesSubplot:xlabel='smoking_status', ylabel='stroke'>

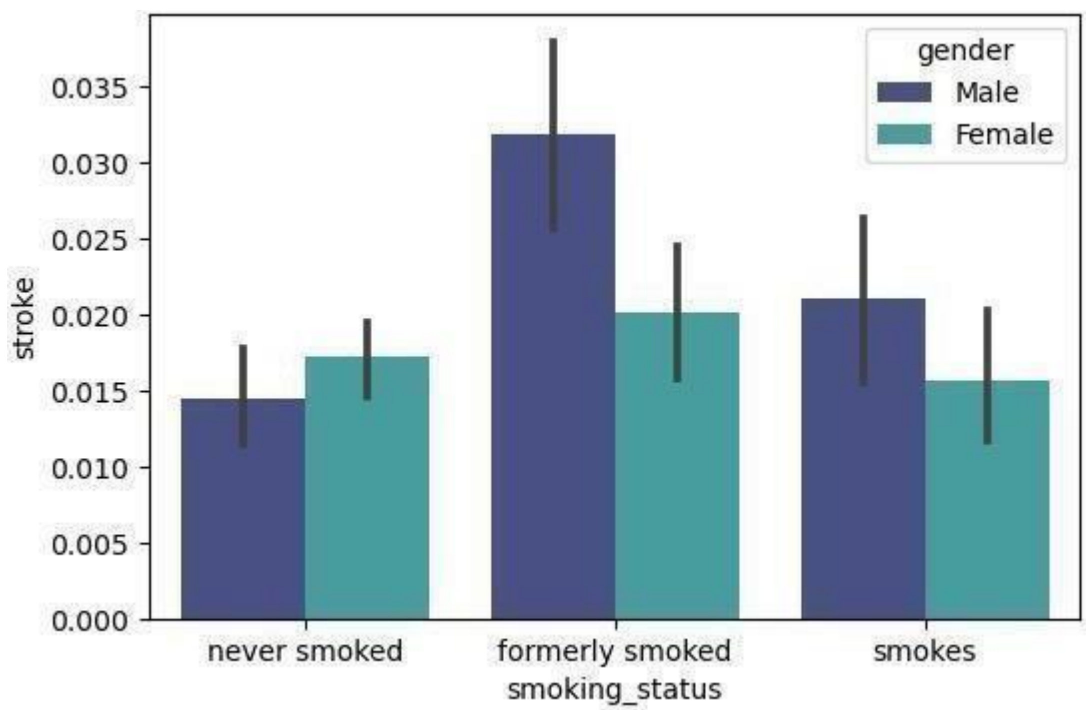


Thus it can be concluded that people who formerly smoked are at greatest risk followed by active smokers and people who have never smoked.

Exploring relationship between stroke and smoking status of patients on the basis of gender

In [26]:
```
plt.figure(dpi=100)
sns.barplot(x='smoking_status' ,y='stroke' ,data=df ,palette='mako' ,hue='gender')
```

Out[26]: <AxesSubplot:xlabel='smoking_status', ylabel='stroke'>

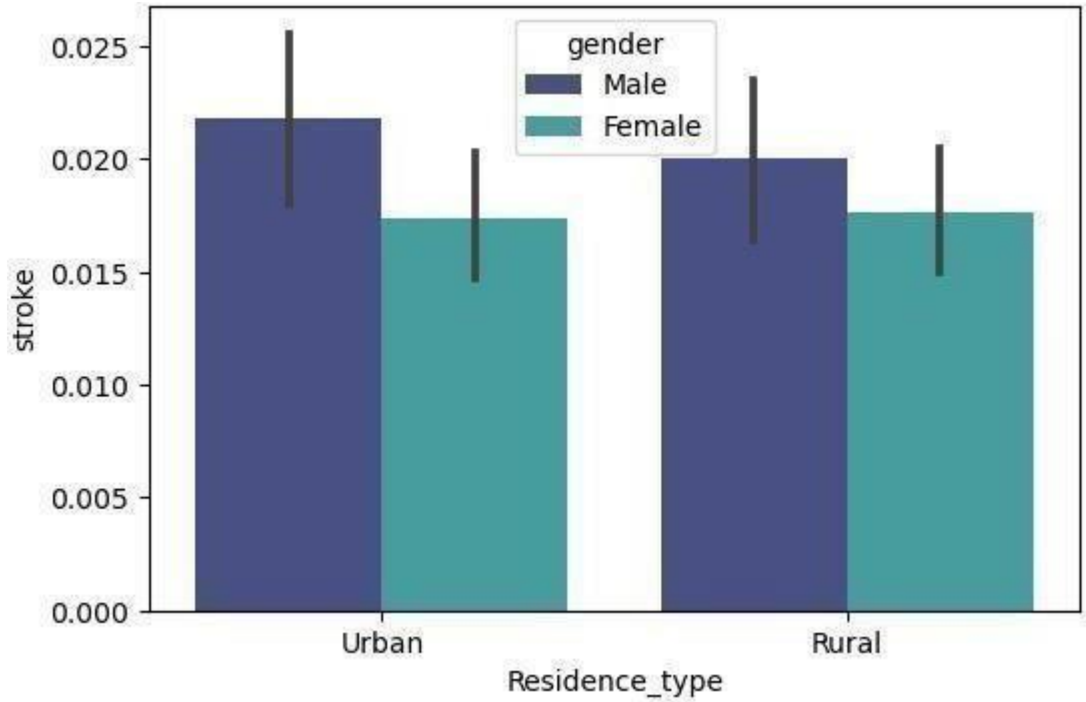


Females at lesser risk than men in general, but in case of never smoked , females are at gretaer risk than men. Probability of stroke is more in former smokers than current active smokers is a notable conclusion

Study between residence type and stroke on the basis of gender

In [27]:
```
plt.figure(dpi=100)
sns.barplot(x='Residence_type', y= 'stroke' , data=df, palette='mako' , hue='gender')
```

Out[27]: <AxesSubplot:xlabel='Residence_type', ylabel='stroke'>

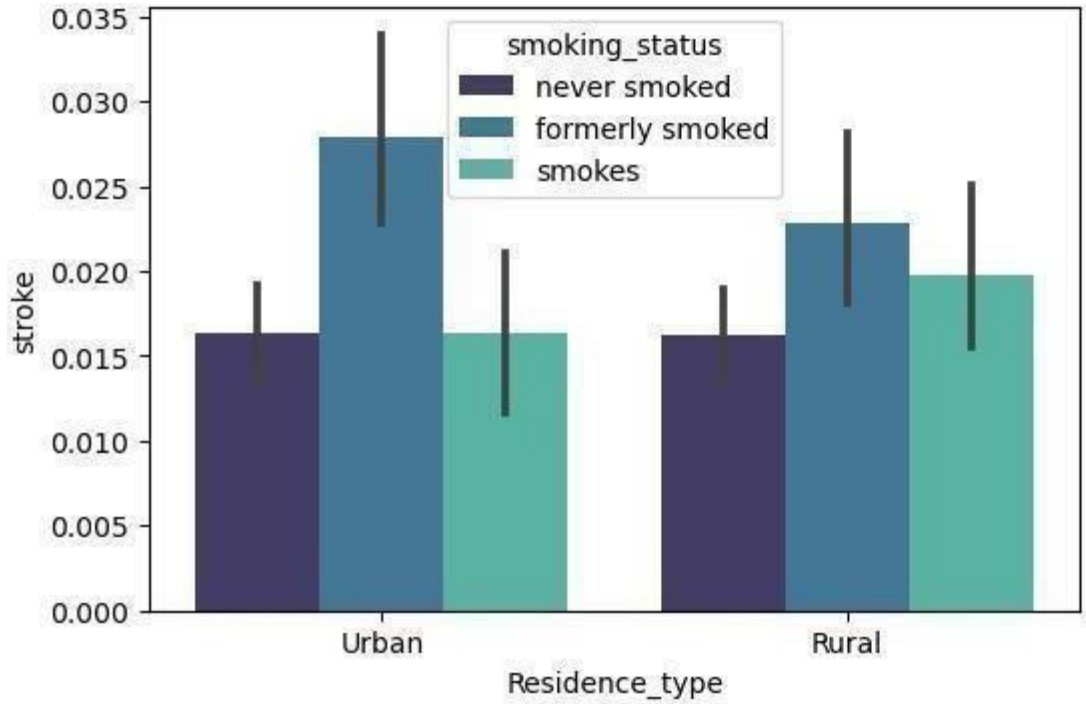


Urban males are at the greatest risk , followed by rural men. However, urban women and rural women have almost the same possibility of stroke

Relation between stroke and residence type on the basis of smoking status

In [28]:
```
plt.figure(dpi=100)
sns.barplot(x='Residence_type', y= 'stroke' , data=df, palette='mako' , hue='smoking_st
```

Out[28]: <AxesSubplot:xlabel='Residence_type', ylabel='stroke'>

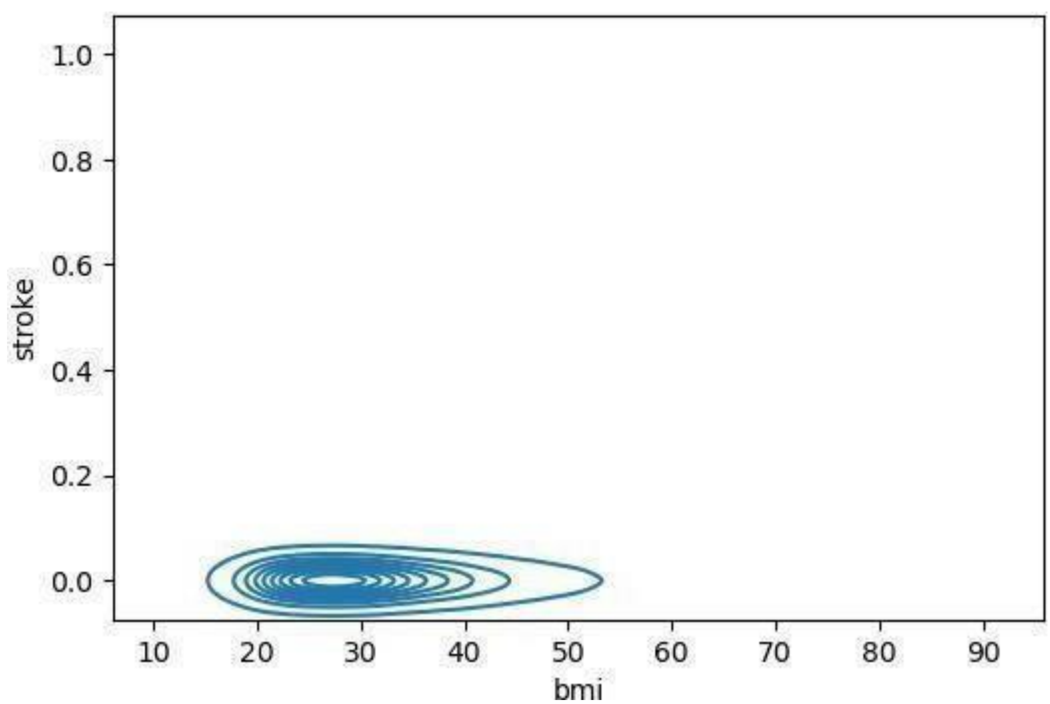


Thisgraphgivesussomeinterestinginsights.Urbanpeoplewhosmokeareatgreatestriskin comparisontotheothercategories.However,thesurprisingfactisthatruralpeoplewhoformerly smokedareatmuchgreaterriskthanurbanpeoplewhocurrentlysmoke.Ruralpeoplewhosmoke are at greater risk than their urban counterparts.

Exploring the relation between bmi and stroke

In [29]:
```
plt.figure(dpi=100)
sns.kdeplot(x='bmi' , y='stroke' , data=df)
```

Out[29]: <AxesSubplot:xlabel='bmi', ylabel='stroke'>

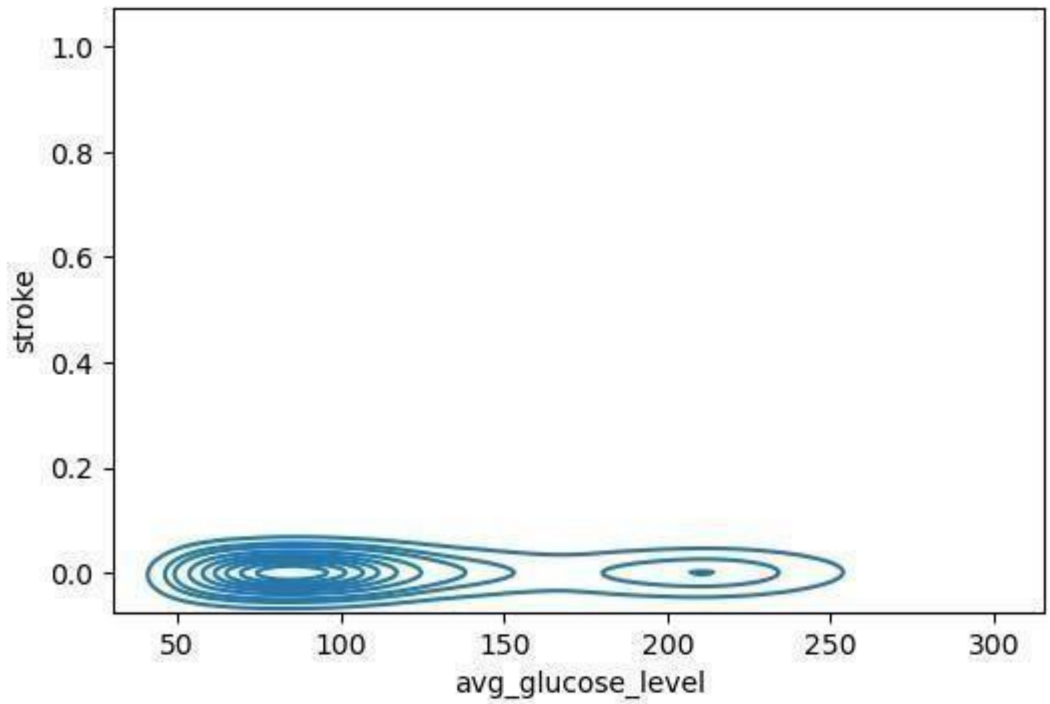


This shows that people having bmi above 27(approx) are the most likely to suffer from stroke

Exploring the relationship between stroke and average glucose level

In [30]:
```
plt.figure(dpi=100)
sns.kdeplot(x="avg_glucose_level" , y="stroke" , data=df)
```

Out[30]: <AxesSubplot:xlabel='avg_glucose_level', ylabel='stroke'>



In [31]:
```
#converting the continuous bmi column to a categorical column and grouping according to
```

In [32]:
```
category=pd.cut(df.bmi , bins=[0,19,25,100] , labels=["underweight" ,"normal"
,"overwei   df.insert(8,"bmi_group",category)
```

In [33]:
```
df.head()
```

Out[33]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_lev
0	Male	58.0	1	0	Yes	Private	Urban	87.9
1	Female	70.0	0	0	Yes	Private	Rural	69.0
2	Female	52.0	0	0	Yes	Private	Urban	77.5
3	Female	75.0	0	1	Yes	Self-employed	Rural	243.5
4	Female	32.0	0	0	Yes	Private	Rural	77.6

In [34]:
```
df.isnull().sum()                #checking for null values in new column
```

Out[34]:
```
gender          0
age             0
```

```
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi_group            0
bmi                  0
smoking_status       0
stroke               0
AGEGROUP             0
combined             0
dtype: int64
```

In [35]:

```
df.sort_values("bmi")
```

Out[35]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucos
17829	Male	39.0	0	0	Yes	Private	Rural	
9435	Female	71.0	0	0	Yes	Govt_job	Rural	
18295	Male	49.0	0	0	Yes	Private	Urban	
1325	Male	40.0	0	0	Yes	Private	Rural	
24784	Female	44.0	0	0	Yes	Private	Urban	
...	...	...	...	...	...	...	...	
484	Female	23.0	1	0	No	Private	Urban	
20539	Female	34.0	0	0	No	Private	Urban	
23306	Female	47.0	1	0	Yes	Private	Urban	
28869	Male	78.0	1	0	Yes	Private	Rural	
2682	Male	38.0	1	0	Yes	Private	Rural	

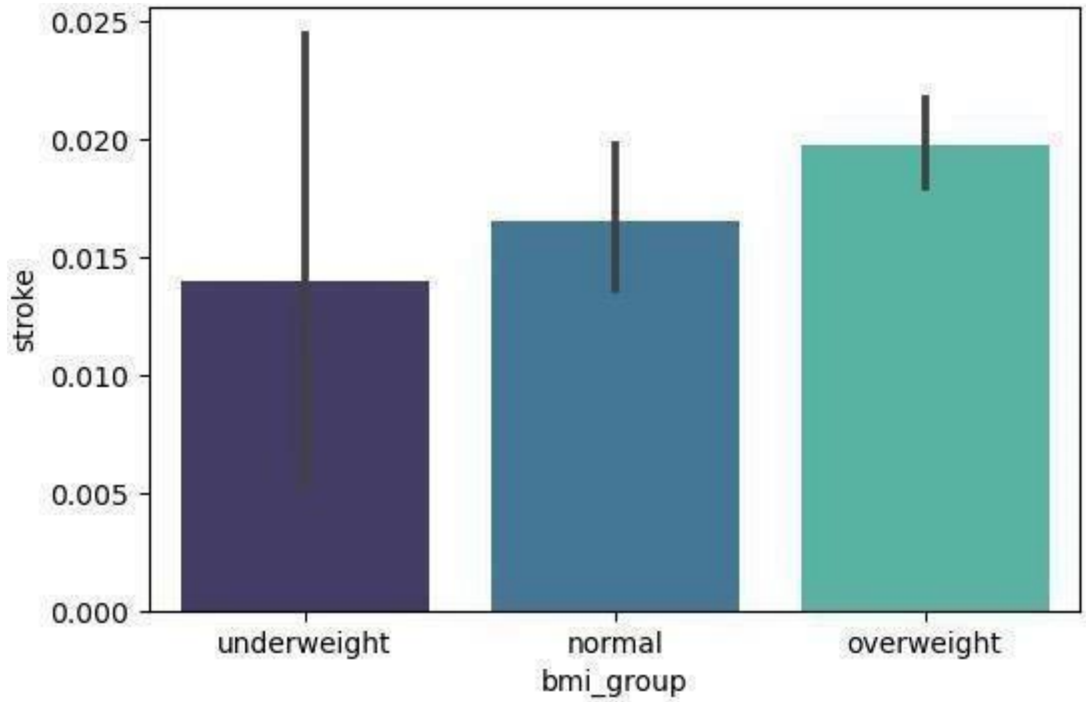
29065 rows × 14 columns

Exploring the relationship between bmi group and stroke

In [41]:

```
plt.figure(dpi=100)
sns.barplot(x="bmi_group" , y="stroke" , data=df , palette="mako")
```

Out[41]: <AxesSubplot:xlabel='bmi_group', ylabel='stroke'>



Fromthegraph it can be concludedthatasbmi increases, probabilityofstrokeincreases.

Converting avg glucose level column from continuous to categorical

In [38]:
```
category2=pd.cut(df.avg_glucose_level , bins=[0,140,200,1000] , labels=['normal'
,'pred   df.insert(9,'glucose_group',category2)
```

In [39]:
```
df.head()
```

Out[39]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_lev
0	Male	58.0	1	0	Yes	Private	Urban	87.9
1	Female	70.0	0	0	Yes	Private	Rural	69.0
2	Female	52.0	0	0	Yes	Private	Urban	77.5
3	Female	75.0	0	1	Yes	Self-employed	Rural	243.5
4	Female	32.0	0	0	Yes	Private	Rural	77.6

In [40]:
```
df.isnull().sum()                          #checking for null values in new column
```

Out[40]:
```
gender              0
age                 0
hypertension        0
heart_disease       0
ever_married        0
work_type           0
Residence_type      0
avg_glucose_level   0
bmi_group           0
```

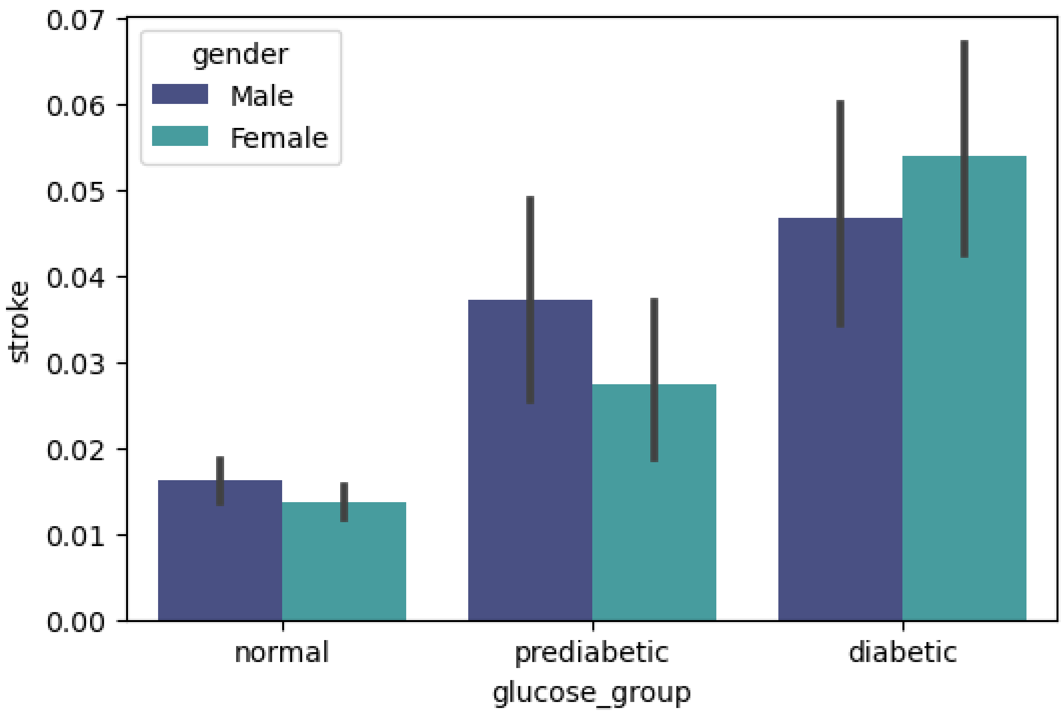
```
glucose_group        0
bmi                  0
smoking_status       0
stroke               0
AGEGROUP             0
combined             0
dtype: int64
```

no null values in new column

Establishing a relationship between avg glucose level groups and stroke

In [43]:
```
plt.figure(dpi=100)
sns.barplot(x='glucose_group', y='stroke', data=df, palette='mako', hue='gender')
```

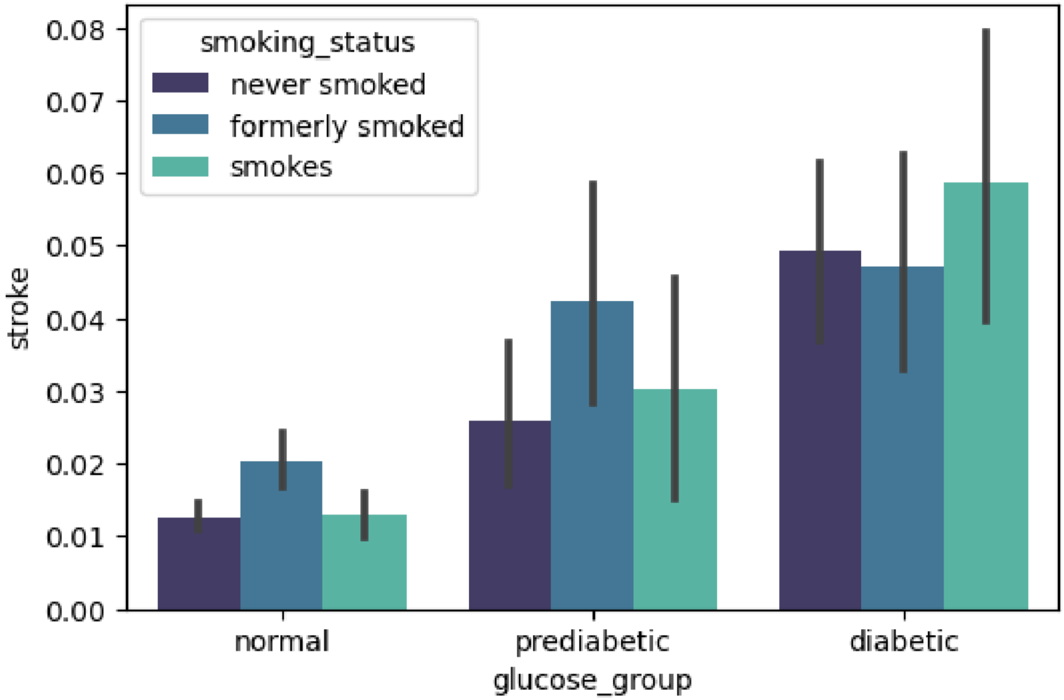
Out[43]: <AxesSubplot:xlabel='glucose_group', ylabel='stroke'>



Probability of stroke highest in diabetic group. In the diabetic group, females are at the greatest risk of stroke. In the prediabetic group, males are at a greater possibility of suffering from a stroke.

In [44]:
```
plt.figure(dpi=100)
sns.barplot(x='glucose_group', y='stroke', data=df, palette='mako', hue='smoking_st
```

Out[44]: <AxesSubplot:xlabel='glucose_group', ylabel='stroke'>


smoking_status
never smoked
formerly smoked
smokes
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
stroke
normal
prediabetic
diabetic
glucose_group

In [ ]:

LOGISTIC REGRESSION- Logistic regression is named for the function used at the core of the method, the logistic function.

The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

1 / (1 + e^-value)

DECISION TREE- **Decision trees** use multiple **algorithms** to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. ... The **decision tree** splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

RANDOM FOREST– Random forests is a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance.

Random forests has a variety of applications, such as recommendation engines, image classification and feature selection. It can be used to classify loyal loan applicants, identify fraudulent activity and predict diseases. It lies at the base of the Boruta algorithm, which selects important features in a dataset.

SUPPORT VECTOR MODEL - A **support vector machine** (SVM) is a supervised **machine** learning **model** that uses classification algorithms for two-group classification problems. After giving an SVM **model** sets of labeled training data for each category, they're able to categorize new text.

GRID SEARCH CV – **GridSearchCV** is a library function that is a member of **sklearn's** model_selection package. It helps to loop through predefined hyper parameters and fit your estimator (model) on your training set. ... In addition to that, you can specify the number of times for the cross-validation for each set of hyperparameters.

In [1]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
df=pd.read_csv('stroke_data.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_lev
0	Male	58.0	1	0	Yes	Private	Urban	87.9
1	Female	70.0	0	0	Yes	Private	Rural	69.0
2	Female	52.0	0	0	Yes	Private	Urban	77.5
3	Female	75.0	0	1	Yes	Self-employed	Rural	243.5
4	Female	32.0	0	0	Yes	Private	Rural	77.6

In [4]:
```
from sklearn.preprocessing import LabelEncoder
```

In [5]:
```
df['gender'].unique()
```

Out[5]:
```
array(['Male', 'Female'], dtype=object)
```

In [6]:
```
label_encoder=LabelEncoder()
```

In [7]:
```
df['gender_label'] = label_encoder.fit_transform(df['gender'])
```

In [8]:
```
df['smoking_label'] = label_encoder.fit_transform(df['smoking_status'])
```

In [9]:
```
df['marriage_label'] = label_encoder.fit_transform(df['ever_married'])
```

In [10]:
```
df['residence_label'] = label_encoder.fit_transform(df['Residence_type'])
```

In [11]:
```
df['work_label'] = label_encoder.fit_transform(df['work_type'])
```

In [12]:
```
df.head()
```

Out[12]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_lev
0	Male	58.0	1	0	Yes	Private	Urban	87.9
1	Female	70.0	0	0	Yes	Private	Rural	69.0
2	Female	52.0	0	0	Yes	Private	Urban	77.5
3	Female	75.0	0	1	Yes	Self-employed	Rural	243.5
4	Female	32.0	0	0	Yes	Private	Rural	77.6

In [13]:
```
df.drop(["gender","smoking_status","ever_married","Residence_type","work_type"],axis=
```
Out[13]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender_label	smoking_label
0	58.0	1	0	87.96	39.2	0	1	1
1	70.0	0	0	69.04	35.9	0	0	0
2	52.0	0	0	77.59	17.7	0	0	0
3	75.0	0	1	243.53	27.0	0	0	1
4	32.0	0	0	77.67	32.3	0	0	2
...	...	...	...	...	...	...	...	...
29060	10.0	0	0	58.64	20.4	0	0	1
29061	56.0	0	0	213.61	55.4	0	0	0
29062	82.0	1	0	91.94	28.9	0	0	0
29063	40.0	0	0	99.16	33.2	0	1	1
29064	82.0	0	0	79.48	20.6	0	0	1

29065 rows × 11 columns

Feature Selection

In [14]:
```
x = df.drop(["stroke",'gender',"smoking_status","ever_married","Residence_type","work
y = df["stroke"]
```

Train Test Split

In [15]:
```
from sklearn.model_selection import train_test_split
```
In [16]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state
```
In [17]:
```
x_train[:5]
```
Out[17]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	gender_label	smoking_label	marriage
5112	79.0	1	0	66.83	19.8	0	1	
13166	32.0	0	0	110.63	33.1	1	1	
5197	64.0	0	0	109.51	25.4	0	1	
25891	24.0	0	0	95.93	23.6	1	2	
755	56.0	0	1	64.66	26.7	0	0	

In [18]:
```
y_train[:5]
```

Out[18]:
```
5112     0
13166    0
5197     0
25891    0
755      0
Name: stroke, dtype: int64
```

Logistic Regression

Model

In [19]:
```
from sklearn.linear_model import LogisticRegression
```

In [20]:
```
log_model = LogisticRegression(max_iter=1000)
```

In [21]:
```
log_model.fit(x_train, y_train)
```

Out[21]:
```
LogisticRegression(max_iter=1000)
```

In [22]:
```
log_pred = log_model.predict(x_test)
```

Accuracy

In [23]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [24]:
```
print(confusion_matrix(y_test, log_pred))
```

```
[[8547    0]
 [ 173    0]]
```

In [25]:
```
print(classification_report(y_test, log_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      8547
           1       0.00      0.00      0.00       173

    accuracy                           0.98      8720
   macro avg       0.49      0.50      0.49      8720
weighted avg       0.96      0.98      0.97      8720
```

```
D:\Anaconda\lib\site-packages\sklearn\metrics\_classification.py:1221: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Accuracy: correct predictions / total predictions

Precision: true positive / total predicted positive (abilty of model to identify relevant data points)

Recall: true positive / total actual positive (abilty of model to final all relevant cases in dataset)

f1-score: 2 * (precision * recall) / (precision + recall) (harmonic mean of precision and recall)

Decision Tree

In [26]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [27]:
```
dtree_model = DecisionTreeClassifier()
```

In [28]:
```
dtree_model.fit(x_train, y_train)
```

Out[28]:
```
DecisionTreeClassifier()
```

In [29]:
```
dtree_pred = dtree_model.predict(x_test)
```

In [30]:
```
print(confusion_matrix(y_test, dtree_pred))
```

```
[[8337  210]
 [ 165    8]]
```

In [31]:
```
print(classification_report(y_test, dtree_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      8547
           1       0.04      0.05      0.04       173

    accuracy                           0.96      8720
   macro avg       0.51      0.51      0.51      8720
weighted avg       0.96      0.96      0.96      8720
```

Random Forest

In [32]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [33]:
```
rfc_model = RandomForestClassifier(n_estimators=1000)
```

In [34]:
```
rfc_model.fit(x_train, y_train)
```

Out[34]:
```
RandomForestClassifier(n_estimators=1000)
```

In [35]:
```
rfc_pred = rfc_model.predict(x_test)
```

In [36]:
```
print(confusion_matrix(y_test, rfc_pred))
```

```
[[8546    1]
 [ 173    0]]
```

In [37]:
```
print(classification_report(y_test, rfc_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      8547
           1       0.00      0.00      0.00       173

    accuracy                           0.98      8720
   macro avg       0.49      0.50      0.49      8720
weighted avg       0.96      0.98      0.97      8720
```

In [38]:
```
from sklearn.metrics import accuracy_score
```

In [39]:
```
accuracy_score(y_test, rfc_pred)
```

Out[39]: 0.9800458715596331

In [40]:
```
ar = [LogisticRegression(), DecisionTreeClassifier(), RandomForestClassifier(n_estima

for i in ar:
    i.fit(x_train, y_train)
    pred = i.predict(x_test)
    print(i, "->", accuracy_score(y_test, pred))
```

```
D:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:762: ConvergenceWarn
ing: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-lear
n.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (h
ttps://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(

LogisticRegression() -> 0.9801605504587156
DecisionTreeClassifier() -> 0.9592889908256881
RandomForestClassifier(n_estimators=1000) -> 0.9800458715596331
```

K-Nearest Neighbors

In [41]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [42]:
```
knn_model = KNeighborsClassifier(n_neighbors=5)
```

In [43]:
```python
knn_model.fit(x_train, y_train)
```
Out[43]: KNeighborsClassifier()

In [44]:
```python
knn_pred = knn_model.predict(x_test)
```

In [45]:
```python
print(confusion_matrix(y_test, knn_pred))
```
```
[[8545    2]
 [ 172    1]]
```

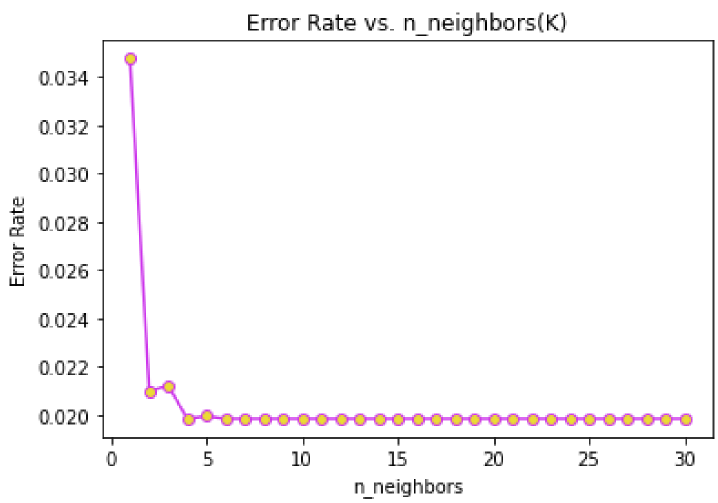
In [46]:
```python
print(classification_report(y_test, knn_pred))
```
```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      8547
           1       0.33      0.01      0.01       173

    accuracy                           0.98      8720
   macro avg       0.66      0.50      0.50      8720
weighted avg       0.97      0.98      0.97      8720
```

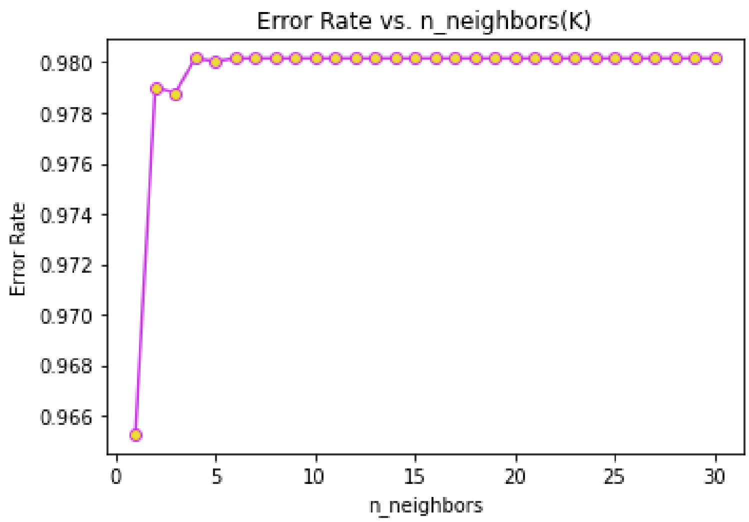
In [47]:
```python
error = []
for i in range(1, 31):
    knn_model = KNeighborsClassifier(n_neighbors=i)
    knn_model.fit(x_train, y_train)
    knn_pred = knn_model.predict(x_test)
    error.append(np.mean(knn_pred != y_test))
```

In [48]:
```python
plt.plot(range(1, 31), error, color="#cc34eb", linestyle="solid", marker="o",markerf
plt.title("Error Rate vs. n_neighbors(K)")
plt.xlabel("n_neighbors")
plt.ylabel("Error Rate")
plt.show()
```



In [49]:
```
error = []
for i in range(1, 31):
    knn_model = KNeighborsClassifier(n_neighbors=i)
    knn_model.fit(x_train, y_train)
    knn_pred = knn_model.predict(x_test)
    error.append(np.mean(accuracy_score(y_test, knn_pred)))

plt.plot(range(1, 31), error,color="#cc34eb", linestyle="solid", marker="o",markerfa
plt.title("Error Rate vs. n_neighbors(K)")
plt.xlabel("n_neighbors")
plt.ylabel("Error Rate")
plt.show()
```



In [50]:
```
knn_model = KNeighborsClassifier(n_neighbors=7)
knn_model.fit(x_train, y_train)
knn_pred = knn_model.predict(x_test)
```

In [51]:
```
print(confusion_matrix(y_test, knn_pred))
```

```
[[8547    0]
 [ 173    0]]
```

In [52]:
```
print(classification_report(y_test, knn_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      8547
           1       0.00      0.00      0.00       173

    accuracy                           0.98      8720
   macro avg       0.49      0.50      0.49      8720
weighted avg       0.96      0.98      0.97      8720
```

```
D:\Anaconda\lib\site-packages\sklearn\metrics\_classification.py:1221: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Support Vector Classifier

In [53]:
```
from sklearn.svm import SVC
```

In [54]:
```
sv_model = SVC()
```

In [55]:
```
sv_model.fit(x_train, y_train)
```

Out[55]: SVC()

In [56]:
```
sv_pred = sv_model.predict(x_test)
```

In [57]:
```
print(confusion_matrix(y_test, sv_pred))
```

```
[[8547    0]
 [ 173    0]]
```

In [58]:
```
print(classification_report(y_test, sv_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      8547
           1       0.00      0.00      0.00       173

    accuracy                           0.98      8720
   macro avg       0.49      0.50      0.49      8720
weighted avg       0.96      0.98      0.97      8720
```

```
D:\Anaconda\lib\site-packages\sklearn\metrics\_classification.py:1221: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Final Model

In [60]:
```
knn_model = KNeighborsClassifier(n_neighbors=7)
knn_model.fit(x, y)
knn_pred = knn_model.predict(x)
```

In [61]:
```
print(confusion_matrix(y, knn_pred))
```

```
[[28517     0]
 [  547     1]]
```

In [62]:
```
print(classification_report(y, knn_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99     28517
           1       1.00      0.00      0.00       548

    accuracy                           0.98     29065
   macro avg       0.99      0.50      0.50     29065
weighted avg       0.98      0.98      0.97     29065
```

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
df=pd.read_csv("stroke_data.csv")
df.head()
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_gluc
0	Male	58.0	1	0	Yes	Private	Urban	
1	Female	70.0	0	0	Yes	Private	Rural	
2	Female	52.0	0	0	Yes	Private	Urban	
3	Female	75.0	0	1	Yes	Self-employed	Rural	
4	Female	32.0	0	0	Yes	Private	Rural	

Label Encoding Categorical Columns

```
from sklearn.preprocessing import LabelEncoder
label_encoder=LabelEncoder()



df['gender_label']=label_encoder.fit_transform(df['gender'])
df['smoking_label']=label_encoder.fit_transform(df['smoking_status'])
df['marriage_label']=label_encoder.fit_transform(df['ever_married'])
df['residence_label']=label_encoder.fit_transform(df['Residence_type'])
df['work_label']=label_encoder.fit_transform(df['work_type'])



df.drop('ever_married',axis=1,inplace=True)
df.drop('work_type',axis=1,inplace=True)
df.drop('Residence_type',axis=1,inplace=True)
df.drop('gender',axis=1,inplace=True)
df.drop('smoking_status',axis=1,inplace=True)


df.head()
```

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender_label	smoking_la
0	58.0	1	0	87.96	39.2	0	1	
1	70.0	0	0	69.04	35.9	0	0	
2	52.0	0	0	77.59	17.7	0	0	
3	75.0	0	1	243.53	27.0	0	0	
4	32.0	0	0	77.67	32.3	0	0	

TRAINING AND TESTING SET DIVISION

```
x=df.drop('stroke',axis=1)
y=df['stroke']
```

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=2021)
```

```
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.svm import SVC
```

```
params={
'C':[0.01,0.1,1,10,100],
'gamma':[0.01,0.1,1,10,100],
'kernel':['rbf']
}
```

GRID SEARCH CV

```
grid=GridSearchCV(SVC(),params,refit=True,verbose=2)
```

```
grid.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
[CV] C=0.01, gamma=0.01, kernel=rbf ..................................
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV] ................... C=0.01, gamma=0.01, kernel=rbf, total=   0.8s
[CV] C=0.01, gamma=0.01, kernel=rbf ..................................
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.8s remaining:    0.0s
[CV] ................... C=0.01, gamma=0.01, kernel=rbf, total=   0.8s
[CV] C=0.01, gamma=0.01, kernel=rbf ..................................
[CV] ................... C=0.01, gamma=0.01, kernel=rbf, total=   0.7s
[CV] C=0.01, gamma=0.01, kernel=rbf ..................................
[CV] ................... C=0.01, gamma=0.01, kernel=rbf, total=   0.8s
[CV] C=0.01, gamma=0.01, kernel=rbf ..................................
[CV] ................... C=0.01, gamma=0.01, kernel=rbf, total=   0.8s
[CV] C=0.01, gamma=0.1, kernel=rbf ...................................
[CV] .................... C=0.01, gamma=0.1, kernel=rbf, total=   1.4s
[CV] C=0.01, gamma=0.1, kernel=rbf ...................................
[CV] .................... C=0.01, gamma=0.1, kernel=rbf, total=   1.4s
[CV] C=0.01, gamma=0.1, kernel=rbf ...................................
[CV] .................... C=0.01, gamma=0.1, kernel=rbf, total=   1.4s
[CV] C=0.01, gamma=0.1, kernel=rbf ...................................
[CV] .................... C=0.01, gamma=0.1, kernel=rbf, total=   1.4s
[CV] C=0.01, gamma=0.1, kernel=rbf ...................................
[CV] .................... C=0.01, gamma=0.1, kernel=rbf, total=   1.4s
[CV] C=0.01, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.01, gamma=1, kernel=rbf, total=   4.8s
[CV] C=0.01, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.01, gamma=1, kernel=rbf, total=   4.8s
[CV] C=0.01, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.01, gamma=1, kernel=rbf, total=   4.8s
[CV] C=0.01, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.01, gamma=1, kernel=rbf, total=   4.8s
[CV] C=0.01, gamma=1, kernel=rbf .....................................
[CV] ...................... C=0.01, gamma=1, kernel=rbf, total=   4.9s
```

```
[CV] C=0.01, gamma=10, kernel=rbf ....................................
[CV] ..................... C=0.01, gamma=10, kernel=rbf, total=   4.1s
[CV] C=0.01, gamma=10, kernel=rbf ....................................
[CV] ..................... C=0.01, gamma=10, kernel=rbf, total=   4.1s
[CV] C=0.01, gamma=10, kernel=rbf ....................................
[CV] ..................... C=0.01, gamma=10, kernel=rbf, total=   4.1s
[CV] C=0.01, gamma=10, kernel=rbf ....................................
[CV] ..................... C=0.01, gamma=10, kernel=rbf, total=   4.1s
[CV] C=0.01, gamma=10, kernel=rbf ....................................
[CV] ..................... C=0.01, gamma=10, kernel=rbf, total=   4.1s
[CV] C=0.01, gamma=100, kernel=rbf ...................................
[CV] .................... C=0.01, gamma=100, kernel=rbf, total=   3.8s
[CV] C=0.01, gamma=100, kernel=rbf ...................................
[CV] .................... C=0.01, gamma=100, kernel=rbf, total=   3.9s
[CV] C=0.01, gamma=100, kernel=rbf ...................................
[CV] .................... C=0.01, gamma=100, kernel=rbf, total=   3.8s
[CV] C=0.01, gamma=100, kernel=rbf ...................................
[CV] .................... C=0.01, gamma=100, kernel=rbf, total=   3.8s
[CV] C=0.01, gamma=100, kernel=rbf ...................................
[CV] .................... C=0.01, gamma=100, kernel=rbf, total=   3.9s
[CV] C=0.1, gamma=0.01, kernel=rbf ...................................
[CV] ..................... C=0.1, gamma=0.01, kernel=rbf, total=   1.8s
[CV] C=0.1, gamma=0.01, kernel=rbf ...................................
[CV] ..................... C=0.1, gamma=0.01, kernel=rbf, total=   1.8s
[CV] C=0.1, gamma=0.01, kernel=rbf ...................................
[CV]                       C=0 1  gamma=0 01  kernel=rbf   total=    1 9s
```

```
grid.best_params_
```

```
{'C': 0.01, 'gamma': 0.01, 'kernel': 'rbf'}
```

```
grid_pred=grid.predict(x_test)
```

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
print(confusion_matrix(y_test,grid_pred))
```

```
[[8547    0]
 [ 173    0]]
```

```
print(classification_report(y_test,grid_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      8547
           1       0.00      0.00      0.00       173

    accuracy                           0.98      8720
   macro avg       0.49      0.50      0.49      8720
weighted avg       0.96      0.98      0.97      8720

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetri
  _warn_prf(average, modifier, msg_start, len(result))
```

What is Naive Bayes algorithm?

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Pros:

- It is easy and fast to predict class of test data set. It also perform well in multi class prediction When
- assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.
- It perform well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

Cons:

- If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency". To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.

Applications of Naive Bayes Algorithms

- **Real time Prediction:** Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.
- **Multi class Prediction:** This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.
- **Text classification/ Spam Filtering/ Sentiment Analysis:** Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e- mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)
- **Recommendation System:** Naive Bayes Classifier and collaborative filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not

Naive Bayes

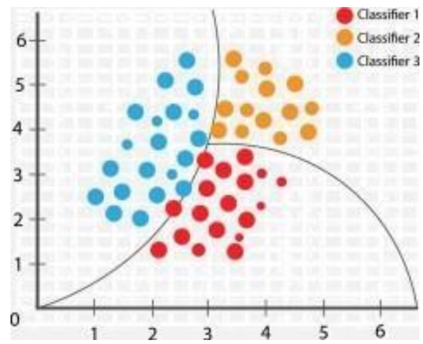
In machine learning, naive Bayes classifiers are a family of simple ”probabilistic classifiers" based on applying Bayes' theorem
with strong (naive) independence assumptions between the features.

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

using Bayesian probability terminology, the above equation can be written as

$$\text{Posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

Naive bayes classified



In [4]:
```python
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
```

In [5]:
```python
df=pd.read_csv('/content/stroke_data.csv')
```

In [6]:
```python
df.head()
```

Out[6]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_lev
0	Male	58.0	1	0	Yes	Private	Urban	87.9
1	Female	70.0	0	0	Yes	Private	Rural	69.0
2	Female	52.0	0	0	Yes	Private	Urban	77.5
3	Female	75.0	0	1	Yes	Self-employed	Rural	243.5
4	Female	32.0	0	0	Yes	Private	Rural	77.6

In [7]:
```python
from sklearn.preprocessing import LabelEncoder
```

In [11]:
```python
label_encoder=LabelEncoder()
```

In [12]:
```python
df['gender_label'] = label_encoder.fit_transform(df['gender'])
df['smoking_label'] = label_encoder.fit_transform(df['smoking_status'])
df['marriage_label'] =
```

In [13]:
```python
df.drop('ever_married', axis=1, inplace=True)
df.drop('work_type', axis=1, inplace=True)
df.drop('Residence_type', axis=1, inplace=True)
df.drop('gender',
```

In [14]:
```python
df.head()
```

Out[14]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender_label	smoking_label	marr
0	58.0	1	0	87.96	39.2	0	1	1	
1	70.0	0	0	69.04	35.9	0	0	0	
2	52.0	0	0	77.59	17.7	0	0	0	
3	75.0	0	1	243.53	27.0	0	0	1	
4	32.0	0	0	77.67	32.3	0	0	2	

In [15]:
```
x=df.drop('stroke', axis=1)
y=df['stroke']
```

In [17]:
```
from sklearn.model_selection import train_test_split
```

In [21]:
```
x_test,x_train,y_test,y_train=train_test_split(x,y,test_size=0.3,random_state=2021)
```

In [22]:
```
from sklearn.naive_bayes import GaussianNB
```

In [23]:
```
bayes_model = GaussianNB()
bayes_model.fit(x_train,y_train)
```

Out[23]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [24]:
```
pred=bayes_model.predict(x_test)
```

In [25]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

In [26]:
```
print(confusion_matrix(y_test, pred))
```

```
[[18680  1290]
 [  273   102]]
```

In [27]:
```
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.94      0.96     19970
           1       0.07      0.27      0.12       375

    accuracy                           0.92     20345
   macro avg       0.53      0.60      0.54     20345
weighted avg       0.97      0.92      0.94     20345
```

In [ ]:

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to fitting linear classifiers and regressors under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just recently in the context of large-scale learning.

SGD has been successfully applied to large-scale and sparse machine learning problems often encountered in text classification and natural language processing. Given that the data is sparse, the classifiers in this module easily scale to problems with more than 10^5 training examples and more than 10^5 features.

Strictly speaking, SGD is merely an optimization technique and does not correspond to a specific family of machine learning models.

The advantages of Stochastic Gradient Descent are:

- Efficiency.
- Ease of implementation (lots of opportunities for code tuning).

The disadvantages of Stochastic Gradient Descent include:

- SGD requires a number of hyper parameters such as the regularization parameter and the number of iterations.
- SGD is sensitive to feature scaling.

In [1]:
```python
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```python
df=pd.read_csv('stroke_data.csv')
```

In [3]:
```python
df.head()
```

Out[3]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_lev
0	Male	58.0	1	0	Yes	Private	Urban	87.9
1	Female	70.0	0	0	Yes	Private	Rural	69.0
2	Female	52.0	0	0	Yes	Private	Urban	77.5
3	Female	75.0	0	1	Yes	Self-employed	Rural	243.5
4	Female	32.0	0	0	Yes	Private	Rural	77.6

In [4]:
```python
from sklearn.preprocessing import LabelEncoder
```

In [5]:
```python
label_encoder=LabelEncoder()
```

In [6]:
```python
df['gender_label'] =
label_encoder.fit_transform(df['gender'])
df['smoking_label'] =
label_encoder.fit_transform(df['smoking_status'])
df['marriage_label'] =
label_encoder.fit_transform(df['ever_married'])
```

In [7]:
```python
df.drop('ever_married', axis=1,
inplace=True) df.drop('work_type',
axis=1, inplace=True)
df.drop('Residence_type', axis=1,
inplace=True) df.drop('gender',
axis=1, inplace=True)
```

In [8]:
```python
df.head()
```

Out[8]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender_label	smoking_label	marr
0	58.0	1	0	87.96	39.2	0	1	1	
1	70.0	0	0	69.04	35.9	0	0	0	
2	52.0	0	0	77.59	17.7	0	0	0	
3	75.0	0	1	243.53	27.0	0	0	1	
4	32.0	0	0	77.67	32.3	0	0	2	

In [9]:
```
x=df.drop('stroke'
,  axis=1)
y=df['stroke']
```

In [10]:
```
from sklearn.model_selection import train_test_split
```

In [11]:
```
x_test,x_train,y_test,y_train=train_test_split(x,y,test_size
=0.3, random_state=2021)
```

In [19]:
```
#data normalization
```

In [20]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

In [22]:
```
standard =
StandardScaler()
standard.fit(x_train)
```

Out[22]:
```
StandardScaler()
```

In [23]:
```
x_train_scaled = standard.transform(x_train)
```

In [24]:
```
x_test_scaled = standard.transform(x_test)
```

In [ ]:
```
```

In [ ]:
```
```

In [12]:
```
from sklearn.linear_model import SGDClassifier
```

In [13]:
```
sgd_model=SGDClassifier()
```

In [25]:
```
sgd_model.fit(x_train_scaled,y_train)
```

Out[25]:
```
SGDClassifier()
```

In [26]:
```
pred=sgd_model.predict(x_test_scaled)
```

In [27]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [37]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [38]:
```
print(confusion_matrix(y_test, pred))
```

```
[[19970     0]
 [  375     0]]
```

In [39]:

```python
from sklearn.metrics import precision_score
```

In [ ]:

```python

```

In [40]:

```python
print(classification_report(y_test, pred))
```

	precision	recall	f1-score	support
0	0.98	1.00	0.99	19970
1	0.00	0.00	0.00	375
accuracy			0.98	20345
macro avg	0.49	0.50	0.50	20345
weighted avg	0.96	0.98	0.97	20345

ARTIFICIAL NEURAL NETWORKS

Neural networks are systems that perform tasks performed by neurons in the human brain. Neural networks include machine learning as part of artificial intelligence (AI) and are the systems in which we develop neurons and brain functionality that replicate the way humans learn.

A neural network (NN) forms a hidden layer that contains units that change the input from output to output so that the output layer can use the value. This transformation is called a neural layer and is called a neural unit. Input to the next level is used by a series of features, called features, which in turn are used as input to the next levels in a series of transformations, each of which has a different value for each level.By repeating these transformations, the neural network learns nonlinear features such as edge shapes, which it then combines with the final layer to make predictions for more complex objects.

Artificial neural networks are biologically inspired computer models modeled on the networks of neurons in the human brain. They can also be seen as learning algorithms that model input-output relationships. Applications of artificial neural networks include pattern recognition and prediction.

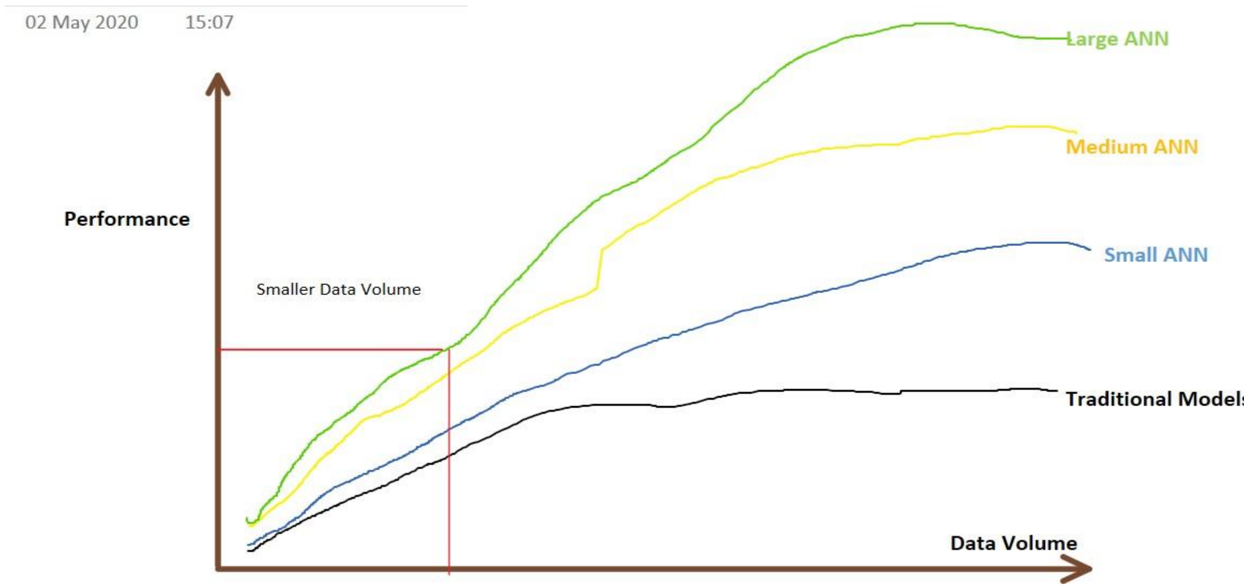
Artificial neural networks (ANNs) are described as machine learning algorithms designed to acquire their own knowledge by extracting useful patterns from data. They apply a nonlinear function to a weighted sum of inputs and model relationships between them. ANNs consist of many interconnected computing units, called neurons, and are functional approximates that map inputs to outputs. ANNs are a function or approximator to map inputs to outputs and vice versa. ANN can model the original neurons of the human brain, so its

processing parts are called “artificial neurons.”

ANN was first introduced in 1943 by the neurophysiologist Warren McCulloch and the mathematician Walter Pitts. However, ANN had its ups and downs.

Post 1990, the advancement in the field of computation (refer to Moore’s law)

followed by the production of powerful GPU cards brought some interest back.



ANNs (Artificial Neural Network) is at the very core of Deep Learning an advanced version of Machine Learning techniques. ANNs are versatile, adaptive, and scalable, making them appropriate to tackle large datasets and highly complex Machine Learning tasks such as image classification (e.g., Google Images), speech recognition (e.g., Apple's Siri), video recommendation (e.g., YouTube), or analyzing sentiments among customers (e.g. Twitter Sentiment Analyzer).

In [99]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [13]:
```
df=pd.read_csv('/stroke_data.csv')
```

In [16]:
```
df.hea
```

Out[16]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_lev
0	Male	58.0	1	0	Yes	Private	Urban	87.9
1	Female	70.0	0	0	Yes	Private	Rural	69.0
2	Female	52.0	0	0	Yes	Private	Urban	77.5
3	Female	75.0	0	1	Yes	Self-employed	Rural	243.5
4	Female	32.0	0	0	Yes	Private	Rural	77.6

In [ ]:

label encoding the categorical column

In [22]:
```
from sklearn.preprocessing import LabelEncoder
```

In [23]:
```
df['gender'].unique()
```

Out[23]:
```
array(['Male', 'Female'], dtype=object)
```

In [24]:
```
label_encoder=LabelEncoder()
```

In [25]:
```
df['gender_label'] = label_encoder.fit_transform(df['gender'])
```

In [26]:
```
df.hea
```

Out[26]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_lev
0	Male	58.0	1	0	Yes	Private	Urban	87.9
1	Female	70.0	0	0	Yes	Private	Rural	69.0
2	Female	52.0	0	0	Yes	Private	Urban	77.5

3	Female	75.0	0	1	Yes	Self-employed	Rural	243.5
4	Female	32.0	0	0	Yes	Private	Rural	77.6

In [27]:
```
df['smoking_label'] = label_encoder.fit_transform(df['smoking_status'])
```

In [28]:
```
df.head()
```

Out[28]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_lev
0	Male	58.0	1	0	Yes	Private	Urban	87.9
1	Female	70.0	0	0	Yes	Private	Rural	69.0
2	Female	52.0	0	0	Yes	Private	Urban	77.5
3	Female	75.0	0	1	Yes	Self-employed	Rural	243.5
4	Female	32.0	0	0	Yes	Private	Rural	77.6

In [29]:
```
df['marriage_label'] = label_encoder.fit_transform(df['ever_married'])
```

In [30]:
```
df['residence_label'] = label_encoder.fit_transform(df['Residence_type'])
```

In [31]:
```
df['work_label'] = label_encoder.fit_transform(df['work_type'])
```

In [35]:
```
df.head()
```

Out[35]:

	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi
0	58.0	1	0	Yes	Private	Urban	87.96	39.2
1	70.0	0	0	Yes	Private	Rural	69.04	35.9
2	52.0	0	0	Yes	Private	Urban	77.59	17.7
3	75.0	0	1	Yes	Self-employed	Rural	243.53	27.0
4	32.0	0	0	Yes	Private	Rural	77.67	32.3

DIVIDING THE DATA INTO TRAINING SET AND TESTING SET

In [69]:
```
df.drop('ever_married', axis=1, inplace=True)
```

In [70]:
```
df.drop('work_type', axis=1, inplace=True)
```

In [71]:
```
df.drop('Residence_type', axis=1, inplace=True)
```

In [74]:
```
df.head()
```

Out[74]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender_label	smoking_label	marr
0	58.0	1	0	87.96	39.2	0	1	1	
1	70.0	0	0	69.04	35.9	0	0	0	
2	52.0	0	0	77.59	17.7	0	0	0	
3	75.0	0	1	243.53	27.0	0	0	1	
4	32.0	0	0	77.67	32.3	0	0	2	

In [75]:
```
x=df.drop('stroke', axis=1)
```

In [76]:
```
y=df['stroke']
```

In [77]:
```
from sklearn.model_selection import train_test_split
```

In [78]:
```
x_test,x_train,y_test,y_train=train_test_split(x,y,test_siz
e=0.3, random_state=2021)
```

In [79]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

In [80]:
```
ann_model = Sequential()
```

In [81]:
```
ann_model.add(Dense(units=30, activation='relu'))    #input
layer
```

In [82]:
```
ann_model.add(Dense(units=15, activation='relu'))    #hidden
layer
```

In [83]:
```
ann_model.add(Dense(units=1, activation='sigmoid'))    #output
layer
```

In [84]:
```
ann_model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

In [85]:
```
ann_model.
fit(
x=x_train,
y=y_trai
n,
epochs=600
,
validation_data=(x_test,
y_test), verbose=1
```

```
Epoch 1/600
273/273 [==============================] - 2s 4ms/step - loss: 1.0367 - accuracy: 0.8494
- val_loss: 0.0969 - val_accuracy: 0.9816
Epoch 2/600
273/273 [==============================] - 1s 3ms/step - loss: 0.0981 - accuracy: 0.9821
- val_loss: 0.1057 - val_accuracy: 0.9816
Epoch 3/600
273/273 [==============================] - 1s 3ms/step - loss: 0.1138 - accuracy: 0.9792
- val_loss: 0.0952 - val_accuracy: 0.9811
Epoch 4/600
273/273 [==============================] - 1s 3ms/step - loss: 0.1178 - accuracy: 0.9755
- val_loss: 0.0949 - val_accuracy: 0.9811
Epoch 5/600
```

273/273 [==============================]- 1s 3ms/step - loss: 0.1047 - accuracy: 0.9817 - val_loss: 0.0883 - val_accuracy: 0.9816
Epoch 6/600
273/273 [==============================]- 1s 3ms/step - loss: 0.1058 - accuracy: 0.9778 - val_loss: 0.0937 - val_accuracy: 0.9816
Epoch 7/600
273/273 [==============================]- 1s 3ms/step - loss: 0.1045 - accuracy: 0.9794 - val_loss: 0.0888 - val_accuracy: 0.9816
Epoch 8/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0966 - accuracy: 0.9819 - val_loss: 0.0886 - val_accuracy: 0.9816
Epoch 9/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0986 - accuracy: 0.9792 - val_loss: 0.0867 - val_accuracy: 0.9816
Epoch 10/600
273/273 [==============================]- 1s 3ms/step - loss: 0.1057 - accuracy: 0.9791 - val_loss: 0.1034 - val_accuracy: 0.9816
Epoch 11/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0966 - accuracy: 0.9808 - val_loss: 0.1017 - val_accuracy: 0.9795
Epoch 12/600
273/273 [==============================]- 1s 3ms/step - loss: 0.1082 - accuracy: 0.9789 - val_loss: 0.0861 - val_accuracy: 0.9816
Epoch 13/600
273/273 [==============================]- 1s 3ms/step - loss: 0.1048 - accuracy: 0.9777 - val_loss: 0.0872 - val_accuracy: 0.9816
Epoch 14/600
273/273 [==============================]- 1s 3ms/step - loss: 0.1042 - accuracy: 0.9791 - val_loss: 0.1118 - val_accuracy: 0.9816
Epoch 15/600
273/273 [==============================]- 1s 3ms/step - loss: 0.1052 - accuracy: 0.9785 - val_loss: 0.0851 - val_accuracy: 0.9816
Epoch 16/600
273/273 [==============================]- 1s 3ms/step - loss: 0.1007 - accuracy: 0.9772 - val_loss: 0.0885 - val_accuracy: 0.9816
Epoch 17/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0873 - accuracy: 0.9811 - val_loss: 0.0846 - val_accuracy: 0.9816
Epoch 18/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0974 - accuracy: 0.9793 - val_loss: 0.0856 - val_accuracy: 0.9816
Epoch 19/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0926 - accuracy: 0.9799 - val_loss: 0.0950 - val_accuracy: 0.9812
Epoch 20/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0971 - accuracy: 0.9785 - val_loss: 0.0956 - val_accuracy: 0.9811
Epoch 21/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0895 - accuracy: 0.9816 - val_loss: 0.0907 - val_accuracy: 0.9810
Epoch 22/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0956 - accuracy: 0.9798 - val_loss: 0.0869 - val_accuracy: 0.9816
Epoch 23/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0914 - accuracy: 0.9797 - val_loss: 0.0858 - val_accuracy: 0.9816
Epoch 24/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0836 - accuracy: 0.9828 - val_loss: 0.0892 - val_accuracy: 0.9816
Epoch 25/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0965 - accuracy: 0.9797 - val_loss: 0.0910 - val_accuracy: 0.9816
Epoch 26/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0984 - accuracy: 0.9781 - val_loss: 0.0852 - val_accuracy: 0.9816

Epoch 27/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0942 - accuracy: 0.9801 - val_loss: 0.0953 - val_accuracy: 0.9816
Epoch 28/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0996 - accuracy: 0.9787 - val_loss: 0.0838 - val_accuracy: 0.9816
Epoch 29/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0931 - accuracy: 0.9793 - val_loss: 0.0850 - val_accuracy: 0.9816
Epoch 30/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0876 - accuracy: 0.9812 - val_loss: 0.0846 - val_accuracy: 0.9816
Epoch 31/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0807 - accuracy: 0.9822 - val_loss: 0.0866 - val_accuracy: 0.9816
Epoch 32/600
273/273 [==============================]- 1s 3ms/step - loss: 0.1011 - accuracy: 0.9778 - val_loss: 0.0960 - val_accuracy: 0.9816
Epoch 33/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0897 - accuracy: 0.9808 - val_loss: 0.0869 - val_accuracy: 0.9816
Epoch 34/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0890 - accuracy: 0.9813 - val_loss: 0.0846 - val_accuracy: 0.9816
Epoch 35/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0883 - accuracy: 0.9803 - val_loss: 0.0871 - val_accuracy: 0.9815
Epoch 36/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0835 - accuracy: 0.9819 - val_loss: 0.0842 - val_accuracy: 0.9816
Epoch 37/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0909 - accuracy: 0.9799 - val_loss: 0.0846 - val_accuracy: 0.9816
Epoch 38/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0834 - accuracy: 0.9818 - val_loss: 0.0863 - val_accuracy: 0.9816
Epoch 39/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0879 - accuracy: 0.9795 - val_loss: 0.0843 - val_accuracy: 0.9816
Epoch 40/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0865 - accuracy: 0.9807 - val_loss: 0.0835 - val_accuracy: 0.9816
Epoch 41/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0854 - accuracy: 0.9805 - val_loss: 0.0884 - val_accuracy: 0.9816
Epoch 42/600
273/273 [==============================]- 1s 3ms/step - loss: 0.1022 - accuracy: 0.9767 - val_loss: 0.0838 - val_accuracy: 0.9816
Epoch 43/600
273/273 [==============================]- 1s 3ms/step - loss: 0.1030 - accuracy: 0.9759 - val_loss: 0.0849 - val_accuracy: 0.9816
Epoch 44/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0898 - accuracy: 0.9797 - val_loss: 0.0868 - val_accuracy: 0.9816
Epoch 45/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0928 - accuracy: 0.9799 - val_loss: 0.0896 - val_accuracy: 0.9814
Epoch 46/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0832 - accuracy: 0.9816 - val_loss: 0.0840 - val_accuracy: 0.9816
Epoch 47/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0848 - accuracy: 0.9817 - val_loss: 0.0863 - val_accuracy: 0.9816
Epoch 48/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0923 - accuracy: 0.9783

- val_loss: 0.0869 - val_accuracy: 0.9816
Epoch 49/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0935 - accuracy: 0.9788
- val_loss: 0.0836 - val_accuracy: 0.9816
Epoch 50/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0828 - accuracy: 0.9817
- val_loss: 0.0896 - val_accuracy: 0.9816
Epoch 51/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0886 - accuracy: 0.9799
- val_loss: 0.0826 - val_accuracy: 0.9816
Epoch 52/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0880 - accuracy: 0.9793
- val_loss: 0.0870 - val_accuracy: 0.9816
Epoch 53/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0899 - accuracy: 0.9787
- val_loss: 0.0877 - val_accuracy: 0.9816
Epoch 54/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0962 - accuracy: 0.9778
- val_loss: 0.0843 - val_accuracy: 0.9816
Epoch 55/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0911 - accuracy: 0.9785
- val_loss: 0.0888 - val_accuracy: 0.9813
Epoch 56/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0866 - accuracy: 0.9809
- val_loss: 0.0832 - val_accuracy: 0.9816
Epoch 57/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0842 - accuracy: 0.9806
- val_loss: 0.0942 - val_accuracy: 0.9813
Epoch 58/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0883 - accuracy: 0.9786
- val_loss: 0.0823 - val_accuracy: 0.9816
Epoch 59/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0903 - accuracy: 0.9792
- val_loss: 0.0875 - val_accuracy: 0.9816
Epoch 60/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0900 - accuracy: 0.9787
- val_loss: 0.0832 - val_accuracy: 0.9816
Epoch 61/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0870 - accuracy: 0.9788
- val_loss: 0.0840 - val_accuracy: 0.9816
Epoch 62/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0807 - accuracy: 0.9823
- val_loss: 0.0834 - val_accuracy: 0.9816
Epoch 63/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0894 - accuracy: 0.9799
- val_loss: 0.0854 - val_accuracy: 0.9816
Epoch 64/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0830 - accuracy: 0.9801
- val_loss: 0.0828 - val_accuracy: 0.9816
Epoch 65/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0825 - accuracy: 0.9806
- val_loss: 0.0862 - val_accuracy: 0.9816
Epoch 66/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0823 - accuracy: 0.9822
- val_loss: 0.0871 - val_accuracy: 0.9816
Epoch 67/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0891 - accuracy: 0.9793
- val_loss: 0.0827 - val_accuracy: 0.9816
Epoch 68/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0885 - accuracy: 0.9792
- val_loss: 0.0833 - val_accuracy: 0.9816
Epoch 69/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0839 - accuracy: 0.9812
- val_loss: 0.0827 - val_accuracy: 0.9816
Epoch 70/600

273/273 [==============================]- 1s 3ms/step - loss: 0.0808 - accuracy: 0.9817 - val_loss: 0.0834 - val_accuracy: 0.9816
Epoch 71/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0855 - accuracy: 0.9801 - val_loss: 0.0865 - val_accuracy: 0.9816
Epoch 72/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0857 - accuracy: 0.9807 - val_loss: 0.0833 - val_accuracy: 0.9816
Epoch 73/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0835 - accuracy: 0.9803 - val_loss: 0.0831 - val_accuracy: 0.9816
Epoch 74/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0857 - accuracy: 0.9793 - val_loss: 0.0866 - val_accuracy: 0.9816
Epoch 75/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0900 - accuracy: 0.9798 - val_loss: 0.0840 - val_accuracy: 0.9816
Epoch 76/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0852 - accuracy: 0.9799 - val_loss: 0.0852 - val_accuracy: 0.9816
Epoch 77/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0912 - accuracy: 0.9778 - val_loss: 0.0888 - val_accuracy: 0.9816
Epoch 78/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0845 - accuracy: 0.9799 - val_loss: 0.0826 - val_accuracy: 0.9816
Epoch 79/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0891 - accuracy: 0.9784 - val_loss: 0.0833 - val_accuracy: 0.9816
Epoch 80/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0841 - accuracy: 0.9808 - val_loss: 0.0843 - val_accuracy: 0.9815
Epoch 81/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0861 - accuracy: 0.9801 - val_loss: 0.0826 - val_accuracy: 0.9816
Epoch 82/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0911 - accuracy: 0.9789 - val_loss: 0.0842 - val_accuracy: 0.9816
Epoch 83/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0916 - accuracy: 0.9781 - val_loss: 0.0832 - val_accuracy: 0.9816
Epoch 84/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0876 - accuracy: 0.9793 - val_loss: 0.0829 - val_accuracy: 0.9816
Epoch 85/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0728 - accuracy: 0.9823 - val_loss: 0.0840 - val_accuracy: 0.9816
Epoch 86/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0827 - accuracy: 0.9808 - val_loss: 0.0828 - val_accuracy: 0.9816
Epoch 87/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0861 - accuracy: 0.9798 - val_loss: 0.0843 - val_accuracy: 0.9816
Epoch 88/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0862 - accuracy: 0.9799 - val_loss: 0.0828 - val_accuracy: 0.9816
Epoch 89/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0860 - accuracy: 0.9793 - val_loss: 0.0826 - val_accuracy: 0.9816
Epoch 90/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0853 - accuracy: 0.9796 - val_loss: 0.0832 - val_accuracy: 0.9816
Epoch 91/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0829 - accuracy: 0.9803 - val_loss: 0.0851 - val_accuracy: 0.9816

```
Epoch 92/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0920 - accuracy: 0.9782
- val_loss: 0.0843 - val_accuracy: 0.9816
Epoch 93/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0901 - accuracy: 0.9782
- val_loss: 0.0861 - val_accuracy: 0.9816
Epoch 94/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0759 - accuracy: 0.9821
- val_loss: 0.0831 - val_accuracy: 0.9816
Epoch 95/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0835 - accuracy: 0.9804
- val_loss: 0.0822 - val_accuracy: 0.9816
Epoch 96/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0873 - accuracy: 0.9795
- val_loss: 0.0829 - val_accuracy: 0.9816
Epoch 97/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0861 - accuracy: 0.9797
- val_loss: 0.0821 - val_accuracy: 0.9816
Epoch 98/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0873 - accuracy: 0.9790
- val_loss: 0.0830 - val_accuracy: 0.9816
Epoch 99/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0829 - accuracy: 0.9809
- val_loss: 0.0847 - val_accuracy: 0.9816
Epoch 100/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0859 - accuracy: 0.9802
- val_loss: 0.0825 - val_accuracy: 0.9816
Epoch 101/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0790 - accuracy: 0.9808
- val_loss: 0.0837 - val_accuracy: 0.9816
Epoch 102/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0919 - accuracy: 0.9778
- val_loss: 0.0830 - val_accuracy: 0.9816
Epoch 103/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0797 - accuracy: 0.9816
- val_loss: 0.0823 - val_accuracy: 0.9816
Epoch 104/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0844 - accuracy: 0.9799
- val_loss: 0.0825 - val_accuracy: 0.9816
Epoch 105/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0780 - accuracy: 0.9816
- val_loss: 0.0825 - val_accuracy: 0.9816
Epoch 106/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0834 - accuracy: 0.9804
- val_loss: 0.0825 - val_accuracy: 0.9816
Epoch 107/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0771 - accuracy: 0.9814
- val_loss: 0.0827 - val_accuracy: 0.9816
Epoch 108/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0704 - accuracy: 0.9835
- val_loss: 0.0823 - val_accuracy: 0.9816
Epoch 109/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0841 - accuracy: 0.9806
- val_loss: 0.0821 - val_accuracy: 0.9816
Epoch 110/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0778 - accuracy: 0.9823
- val_loss: 0.0833 - val_accuracy: 0.9815
Epoch 111/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0798 - accuracy: 0.9812
- val_loss: 0.0831 - val_accuracy: 0.9816
Epoch 112/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0830 - accuracy: 0.9802
- val_loss: 0.0822 - val_accuracy: 0.9816
Epoch 113/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0816 - accuracy: 0.9795
```

- val_loss: 0.0911 - val_accuracy: 0.9816
Epoch 114/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0882 - accuracy: 0.9795 - val_loss: 0.0821 - val_accuracy: 0.9816
Epoch 115/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0897 - accuracy: 0.9777 - val_loss: 0.0815 - val_accuracy: 0.9816
Epoch 116/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0814 - accuracy: 0.9805 - val_loss: 0.0832 - val_accuracy: 0.9816
Epoch 117/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0881 - accuracy: 0.9774 - val_loss: 0.0837 - val_accuracy: 0.9816
Epoch 118/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0807 - accuracy: 0.9807 - val_loss: 0.0865 - val_accuracy: 0.9816
Epoch 119/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0818 - accuracy: 0.9800 - val_loss: 0.0837 - val_accuracy: 0.9816
Epoch 120/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0821 - accuracy: 0.9799 - val_loss: 0.0827 - val_accuracy: 0.9816
Epoch 121/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0817 - accuracy: 0.9796 - val_loss: 0.0829 - val_accuracy: 0.9816
Epoch 122/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0743 - accuracy: 0.9819 - val_loss: 0.0822 - val_accuracy: 0.9816
Epoch 123/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0746 - accuracy: 0.9822 - val_loss: 0.0833 - val_accuracy: 0.9816
Epoch 124/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0845 - accuracy: 0.9790 - val_loss: 0.0823 - val_accuracy: 0.9816
Epoch 125/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0837 - accuracy: 0.9794 - val_loss: 0.0835 - val_accuracy: 0.9816
Epoch 126/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0795 - accuracy: 0.9799 - val_loss: 0.0837 - val_accuracy: 0.9816
Epoch 127/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0855 - accuracy: 0.9795 - val_loss: 0.0843 - val_accuracy: 0.9816
Epoch 128/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0837 - accuracy: 0.9792 - val_loss: 0.0830 - val_accuracy: 0.9816
Epoch 129/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0861 - accuracy: 0.9789 - val_loss: 0.0823 - val_accuracy: 0.9816
Epoch 130/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0773 - accuracy: 0.9808 - val_loss: 0.0834 - val_accuracy: 0.9816
Epoch 131/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0827 - accuracy: 0.9804 - val_loss: 0.0878 - val_accuracy: 0.9816
Epoch 132/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0754 - accuracy: 0.9823 - val_loss: 0.0831 - val_accuracy: 0.9816
Epoch 133/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0764 - accuracy: 0.9820 - val_loss: 0.0828 - val_accuracy: 0.9816
Epoch 134/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0835 - accuracy: 0.9797 - val_loss: 0.0836 - val_accuracy: 0.9816
Epoch 135/600

273/273 [==============================]- 1s 3ms/step - loss: 0.0836 - accuracy: 0.9803 - val_loss: 0.0841 - val_accuracy: 0.9816
Epoch 136/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0707 - accuracy: 0.9836 - val_loss: 0.0846 - val_accuracy: 0.9816
Epoch 137/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0800 - accuracy: 0.9802 - val_loss: 0.0822 - val_accuracy: 0.9816
Epoch 138/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0790 - accuracy: 0.9811 - val_loss: 0.0897 - val_accuracy: 0.9816
Epoch 139/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0866 - accuracy: 0.9791 - val_loss: 0.0827 - val_accuracy: 0.9816
Epoch 140/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0835 - accuracy: 0.9792 - val_loss: 0.0831 - val_accuracy: 0.9816
Epoch 141/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0739 - accuracy: 0.9819 - val_loss: 0.0848 - val_accuracy: 0.9816
Epoch 142/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0840 - accuracy: 0.9789 - val_loss: 0.0830 - val_accuracy: 0.9816
Epoch 143/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0850 - accuracy: 0.9789 - val_loss: 0.0847 - val_accuracy: 0.9816
Epoch 144/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0831 - accuracy: 0.9796 - val_loss: 0.0841 - val_accuracy: 0.9816
Epoch 145/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0834 - accuracy: 0.9792 - val_loss: 0.0831 - val_accuracy: 0.9816
Epoch 146/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0814 - accuracy: 0.9801 - val_loss: 0.0841 - val_accuracy: 0.9816
Epoch 147/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0839 - accuracy: 0.9785 - val_loss: 0.0869 - val_accuracy: 0.9816
Epoch 148/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0813 - accuracy: 0.9790 - val_loss: 0.0843 - val_accuracy: 0.9816
Epoch 149/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0759 - accuracy: 0.9817 - val_loss: 0.0840 - val_accuracy: 0.9815
Epoch 150/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0808 - accuracy: 0.9800 - val_loss: 0.0839 - val_accuracy: 0.9816
Epoch 151/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0838 - accuracy: 0.9789 - val_loss: 0.0847 - val_accuracy: 0.9815
Epoch 152/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0833 - accuracy: 0.9800 - val_loss: 0.0825 - val_accuracy: 0.9816
Epoch 153/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0812 - accuracy: 0.9798 - val_loss: 0.0825 - val_accuracy: 0.9816
Epoch 154/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0798 - accuracy: 0.9804 - val_loss: 0.0840 - val_accuracy: 0.9816
Epoch 155/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0782 - accuracy: 0.9812 - val_loss: 0.0839 - val_accuracy: 0.9816
Epoch 156/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0875 - accuracy: 0.9769 - val_loss: 0.0833 - val_accuracy: 0.9816

Epoch 157/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0754 - accuracy: 0.9811 - val_loss: 0.0832 - val_accuracy: 0.9811
Epoch 158/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0751 - accuracy: 0.9822 - val_loss: 0.0878 - val_accuracy: 0.9815
Epoch 159/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0770 - accuracy: 0.9816 - val_loss: 0.0856 - val_accuracy: 0.9815
Epoch 160/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0806 - accuracy: 0.9806 - val_loss: 0.0836 - val_accuracy: 0.9815
Epoch 161/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0754 - accuracy: 0.9809 - val_loss: 0.0833 - val_accuracy: 0.9816
Epoch 162/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0762 - accuracy: 0.9811 - val_loss: 0.0850 - val_accuracy: 0.9816
Epoch 163/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0783 - accuracy: 0.9807 - val_loss: 0.0843 - val_accuracy: 0.9816
Epoch 164/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0788 - accuracy: 0.9808 - val_loss: 0.0841 - val_accuracy: 0.9815
Epoch 165/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0788 - accuracy: 0.9805 - val_loss: 0.0831 - val_accuracy: 0.9814
Epoch 166/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0726 - accuracy: 0.9820 - val_loss: 0.0890 - val_accuracy: 0.9812
Epoch 167/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0847 - accuracy: 0.9790 - val_loss: 0.0861 - val_accuracy: 0.9816
Epoch 168/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0756 - accuracy: 0.9814 - val_loss: 0.0831 - val_accuracy: 0.9815
Epoch 169/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0786 - accuracy: 0.9798 - val_loss: 0.0846 - val_accuracy: 0.9816
Epoch 170/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0839 - accuracy: 0.9787 - val_loss: 0.0866 - val_accuracy: 0.9816
Epoch 171/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0738 - accuracy: 0.9822 - val_loss: 0.0838 - val_accuracy: 0.9815
Epoch 172/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0863 - accuracy: 0.9779 - val_loss: 0.0832 - val_accuracy: 0.9815
Epoch 173/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0877 - accuracy: 0.9772 - val_loss: 0.0840 - val_accuracy: 0.9816
Epoch 174/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0849 - accuracy: 0.9783 - val_loss: 0.0835 - val_accuracy: 0.9816
Epoch 175/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0732 - accuracy: 0.9814 - val_loss: 0.0846 - val_accuracy: 0.9810
Epoch 176/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0878 - accuracy: 0.9777 - val_loss: 0.0836 - val_accuracy: 0.9815
Epoch 177/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0717 - accuracy: 0.9825 - val_loss: 0.0865 - val_accuracy: 0.9814
Epoch 178/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0878 - accuracy: 0.9782

- val_loss: 0.0870 - val_accuracy: 0.9816
Epoch 179/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0804 - accuracy: 0.9793
- val_loss: 0.0838 - val_accuracy: 0.9814
Epoch 180/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0775 - accuracy: 0.9794
- val_loss: 0.0844 - val_accuracy: 0.9815
Epoch 181/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0776 - accuracy: 0.9800
- val_loss: 0.0831 - val_accuracy: 0.9813
Epoch 182/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0732 - accuracy: 0.9820
- val_loss: 0.0856 - val_accuracy: 0.9816
Epoch 183/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0791 - accuracy: 0.9798
- val_loss: 0.0843 - val_accuracy: 0.9810
Epoch 184/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0779 - accuracy: 0.9797
- val_loss: 0.0873 - val_accuracy: 0.9811
Epoch 185/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0773 - accuracy: 0.9813
- val_loss: 0.0839 - val_accuracy: 0.9816
Epoch 186/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0843 - accuracy: 0.9791
- val_loss: 0.0836 - val_accuracy: 0.9815
Epoch 187/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0818 - accuracy: 0.9789
- val_loss: 0.0850 - val_accuracy: 0.9815
Epoch 188/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0753 - accuracy: 0.9817
- val_loss: 0.0838 - val_accuracy: 0.9816
Epoch 189/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0819 - accuracy: 0.9785
- val_loss: 0.0843 - val_accuracy: 0.9815
Epoch 190/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0694 - accuracy: 0.9832
- val_loss: 0.0845 - val_accuracy: 0.9811
Epoch 191/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0776 - accuracy: 0.9804
- val_loss: 0.0846 - val_accuracy: 0.9816
Epoch 192/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0788 - accuracy: 0.9804
- val_loss: 0.0854 - val_accuracy: 0.9816
Epoch 193/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0811 - accuracy: 0.9797
- val_loss: 0.0856 - val_accuracy: 0.9815
Epoch 194/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0748 - accuracy: 0.9810
- val_loss: 0.0852 - val_accuracy: 0.9816
Epoch 195/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0746 - accuracy: 0.9819
- val_loss: 0.0862 - val_accuracy: 0.9815
Epoch 196/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0737 - accuracy: 0.9809
- val_loss: 0.0841 - val_accuracy: 0.9816
Epoch 197/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0749 - accuracy: 0.9817
- val_loss: 0.0843 - val_accuracy: 0.9813
Epoch 198/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0827 - accuracy: 0.9800
- val_loss: 0.0848 - val_accuracy: 0.9811
Epoch 199/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0789 - accuracy: 0.9799
- val_loss: 0.0844 - val_accuracy: 0.9814
Epoch 200/600

273/273 [==============================]- 1s 3ms/step - loss: 0.0806 - accuracy: 0.9779 - val_loss: 0.0846 - val_accuracy: 0.9816
Epoch 201/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0819 - accuracy: 0.9790 - val_loss: 0.0839 - val_accuracy: 0.9816
Epoch 202/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0869 - accuracy: 0.9781 - val_loss: 0.0853 - val_accuracy: 0.9814
Epoch 203/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0725 - accuracy: 0.9813 - val_loss: 0.0867 - val_accuracy: 0.9816
Epoch 204/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0754 - accuracy: 0.9813 - val_loss: 0.0844 - val_accuracy: 0.9815
Epoch 205/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0810 - accuracy: 0.9789 - val_loss: 0.0854 - val_accuracy: 0.9816
Epoch 206/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0809 - accuracy: 0.9792 - val_loss: 0.0853 - val_accuracy: 0.9814
Epoch 207/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0751 - accuracy: 0.9803 - val_loss: 0.0862 - val_accuracy: 0.9810
Epoch 208/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0734 - accuracy: 0.9819 - val_loss: 0.0855 - val_accuracy: 0.9809
Epoch 209/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0790 - accuracy: 0.9797 - val_loss: 0.0846 - val_accuracy: 0.9815
Epoch 210/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0798 - accuracy: 0.9795 - val_loss: 0.0928 - val_accuracy: 0.9816
Epoch 211/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0741 - accuracy: 0.9813 - val_loss: 0.0882 - val_accuracy: 0.9815
Epoch 212/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0837 - accuracy: 0.9776 - val_loss: 0.0872 - val_accuracy: 0.9815
Epoch 213/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0795 - accuracy: 0.9802 - val_loss: 0.0861 - val_accuracy: 0.9811
Epoch 214/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0751 - accuracy: 0.9816 - val_loss: 0.0848 - val_accuracy: 0.9815
Epoch 215/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0794 - accuracy: 0.9788 - val_loss: 0.0868 - val_accuracy: 0.9814
Epoch 216/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0796 - accuracy: 0.9805 - val_loss: 0.0865 - val_accuracy: 0.9815
Epoch 217/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0827 - accuracy: 0.9787 - val_loss: 0.0886 - val_accuracy: 0.9816
Epoch 218/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0692 - accuracy: 0.9826 - val_loss: 0.0873 - val_accuracy: 0.9816
Epoch 219/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0825 - accuracy: 0.9792 - val_loss: 0.0884 - val_accuracy: 0.9816
Epoch 220/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0801 - accuracy: 0.9800 - val_loss: 0.0867 - val_accuracy: 0.9815
Epoch 221/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0800 - accuracy: 0.9793 - val_loss: 0.0875 - val_accuracy: 0.9816

Epoch 222/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0769 - accuracy: 0.9810
- val_loss: 0.0853 - val_accuracy: 0.9815
Epoch 223/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0773 - accuracy: 0.9809
- val_loss: 0.0858 - val_accuracy: 0.9814
Epoch 224/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0752 - accuracy: 0.9809
- val_loss: 0.0868 - val_accuracy: 0.9815
Epoch 225/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0782 - accuracy: 0.9795
- val_loss: 0.0871 - val_accuracy: 0.9815
Epoch 226/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0735 - accuracy: 0.9813
- val_loss: 0.0864 - val_accuracy: 0.9815
Epoch 227/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0742 - accuracy: 0.9811
- val_loss: 0.0871 - val_accuracy: 0.9815
Epoch 228/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0735 - accuracy: 0.9812
- val_loss: 0.0883 - val_accuracy: 0.9810
Epoch 229/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0810 - accuracy: 0.9786
- val_loss: 0.0853 - val_accuracy: 0.9815
Epoch 230/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0711 - accuracy: 0.9822
- val_loss: 0.0869 - val_accuracy: 0.9815
Epoch 231/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0804 - accuracy: 0.9793
- val_loss: 0.0874 - val_accuracy: 0.9814
Epoch 232/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0740 - accuracy: 0.9822
- val_loss: 0.0863 - val_accuracy: 0.9815
Epoch 233/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0784 - accuracy: 0.9798
- val_loss: 0.0867 - val_accuracy: 0.9815
Epoch 234/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0683 - accuracy: 0.9826
- val_loss: 0.0880 - val_accuracy: 0.9813
Epoch 235/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0711 - accuracy: 0.9825
- val_loss: 0.0937 - val_accuracy: 0.9799
Epoch 236/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0841 - accuracy: 0.9778
- val_loss: 0.0906 - val_accuracy: 0.9816
Epoch 237/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0799 - accuracy: 0.9796
- val_loss: 0.0852 - val_accuracy: 0.9814
Epoch 238/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0705 - accuracy: 0.9822
- val_loss: 0.0868 - val_accuracy: 0.9814
Epoch 239/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0734 - accuracy: 0.9811
- val_loss: 0.0871 - val_accuracy: 0.9812
Epoch 240/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0774 - accuracy: 0.9805
- val_loss: 0.0871 - val_accuracy: 0.9814
Epoch 241/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0680 - accuracy: 0.9825
- val_loss: 0.0867 - val_accuracy: 0.9815
Epoch 242/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0817 - accuracy: 0.9802
- val_loss: 0.0894 - val_accuracy: 0.9810
Epoch 243/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0687 - accuracy: 0.9825

```
- val_loss: 0.0887 - val_accuracy: 0.9814
Epoch 244/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0777 - accuracy: 0.9802
- val_loss: 0.0871 - val_accuracy: 0.9814
Epoch 245/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0713 - accuracy: 0.9824
- val_loss: 0.0894 - val_accuracy: 0.9813
Epoch 246/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0840 - accuracy: 0.9793
- val_loss: 0.0879 - val_accuracy: 0.9814
Epoch 247/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0766 - accuracy: 0.9811
- val_loss: 0.0884 - val_accuracy: 0.9812
Epoch 248/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0688 - accuracy: 0.9833
- val_loss: 0.0872 - val_accuracy: 0.9813
Epoch 249/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0848 - accuracy: 0.9782
- val_loss: 0.0887 - val_accuracy: 0.9815
Epoch 250/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0767 - accuracy: 0.9805
- val_loss: 0.0970 - val_accuracy: 0.9793
Epoch 251/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0760 - accuracy: 0.9814
- val_loss: 0.0892 - val_accuracy: 0.9816
Epoch 252/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0722 - accuracy: 0.9812
- val_loss: 0.0954 - val_accuracy: 0.9801
Epoch 253/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0914 - accuracy: 0.9757
- val_loss: 0.0865 - val_accuracy: 0.9815
Epoch 254/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0768 - accuracy: 0.9800
- val_loss: 0.0866 - val_accuracy: 0.9813
Epoch 255/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0683 - accuracy: 0.9832
- val_loss: 0.0879 - val_accuracy: 0.9815
Epoch 256/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0666 - accuracy: 0.9830
- val_loss: 0.0884 - val_accuracy: 0.9813
Epoch 257/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0785 - accuracy: 0.9790
- val_loss: 0.0907 - val_accuracy: 0.9815
Epoch 258/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0761 - accuracy: 0.9806
- val_loss: 0.0898 - val_accuracy: 0.9815
Epoch 259/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0830 - accuracy: 0.9791
- val_loss: 0.0897 - val_accuracy: 0.9813
Epoch 260/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0774 - accuracy: 0.9811
- val_loss: 0.0900 - val_accuracy: 0.9812
Epoch 261/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0722 - accuracy: 0.9810
- val_loss: 0.0901 - val_accuracy: 0.9812
Epoch 262/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0800 - accuracy: 0.9795
- val_loss: 0.0882 - val_accuracy: 0.9814
Epoch 263/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0793 - accuracy: 0.9785
- val_loss: 0.0882 - val_accuracy: 0.9815
Epoch 264/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0635 - accuracy: 0.9838
- val_loss: 0.0927 - val_accuracy: 0.9814
Epoch 265/600
```

273/273 [==============================]- 1s 4ms/step - loss: 0.0690 - accuracy: 0.9830 - val_loss: 0.0867 - val_accuracy: 0.9814
Epoch 266/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0766 - accuracy: 0.9805 - val_loss: 0.0881 - val_accuracy: 0.9813
Epoch 267/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0700 - accuracy: 0.9824 - val_loss: 0.0884 - val_accuracy: 0.9814
Epoch 268/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0767 - accuracy: 0.9792 - val_loss: 0.0877 - val_accuracy: 0.9813
Epoch 269/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0820 - accuracy: 0.9787 - val_loss: 0.0909 - val_accuracy: 0.9815
Epoch 270/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0686 - accuracy: 0.9819 - val_loss: 0.0883 - val_accuracy: 0.9815
Epoch 271/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0656 - accuracy: 0.9826 - val_loss: 0.0917 - val_accuracy: 0.9815
Epoch 272/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0804 - accuracy: 0.9783 - val_loss: 0.0939 - val_accuracy: 0.9814
Epoch 273/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0817 - accuracy: 0.9777 - val_loss: 0.0894 - val_accuracy: 0.9814
Epoch 274/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0716 - accuracy: 0.9821 - val_loss: 0.0935 - val_accuracy: 0.9813
Epoch 275/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0685 - accuracy: 0.9834 - val_loss: 0.0863 - val_accuracy: 0.9813
Epoch 276/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0797 - accuracy: 0.9797 - val_loss: 0.0905 - val_accuracy: 0.9815
Epoch 277/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0751 - accuracy: 0.9800 - val_loss: 0.0884 - val_accuracy: 0.9816
Epoch 278/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0780 - accuracy: 0.9789 - val_loss: 0.0891 - val_accuracy: 0.9815
Epoch 279/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0836 - accuracy: 0.9775 - val_loss: 0.0908 - val_accuracy: 0.9816
Epoch 280/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0740 - accuracy: 0.9806 - val_loss: 0.0904 - val_accuracy: 0.9815
Epoch 281/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0770 - accuracy: 0.9796 - val_loss: 0.0877 - val_accuracy: 0.9814
Epoch 282/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0803 - accuracy: 0.9806 - val_loss: 0.0905 - val_accuracy: 0.9816
Epoch 283/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0817 - accuracy: 0.9793 - val_loss: 0.0880 - val_accuracy: 0.9816
Epoch 284/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0697 - accuracy: 0.9813 - val_loss: 0.0932 - val_accuracy: 0.9815
Epoch 285/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0831 - accuracy: 0.9791 - val_loss: 0.0870 - val_accuracy: 0.9814
Epoch 286/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0759 - accuracy: 0.9802 - val_loss: 0.0902 - val_accuracy: 0.9814

```
Epoch 287/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0734 - accuracy: 0.9808
- val_loss: 0.0930 - val_accuracy: 0.9816
Epoch 288/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0790 - accuracy: 0.9804
- val_loss: 0.0900 - val_accuracy: 0.9814
Epoch 289/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0801 - accuracy: 0.9777
- val_loss: 0.0931 - val_accuracy: 0.9815
Epoch 290/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0754 - accuracy: 0.9803
- val_loss: 0.0894 - val_accuracy: 0.9814
Epoch 291/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0762 - accuracy: 0.9803
- val_loss: 0.0903 - val_accuracy: 0.9815
Epoch 292/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0816 - accuracy: 0.9794
- val_loss: 0.0938 - val_accuracy: 0.9815
Epoch 293/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0732 - accuracy: 0.9805
- val_loss: 0.0904 - val_accuracy: 0.9816
Epoch 294/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0646 - accuracy: 0.9834
- val_loss: 0.0883 - val_accuracy: 0.9814
Epoch 295/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0772 - accuracy: 0.9794
- val_loss: 0.0882 - val_accuracy: 0.9815
Epoch 296/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0721 - accuracy: 0.9805
- val_loss: 0.0936 - val_accuracy: 0.9808
Epoch 297/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0814 - accuracy: 0.9784
- val_loss: 0.0901 - val_accuracy: 0.9815
Epoch 298/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0714 - accuracy: 0.9815
- val_loss: 0.0938 - val_accuracy: 0.9801
Epoch 299/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0733 - accuracy: 0.9805
- val_loss: 0.0932 - val_accuracy: 0.9815
Epoch 300/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0778 - accuracy: 0.9795
- val_loss: 0.0926 - val_accuracy: 0.9814
Epoch 301/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0754 - accuracy: 0.9805
- val_loss: 0.0910 - val_accuracy: 0.9814
Epoch 302/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0704 - accuracy: 0.9823
- val_loss: 0.0899 - val_accuracy: 0.9815
Epoch 303/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0815 - accuracy: 0.9780
- val_loss: 0.0940 - val_accuracy: 0.9816
Epoch 304/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0782 - accuracy: 0.9790
- val_loss: 0.0935 - val_accuracy: 0.9814
Epoch 305/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0790 - accuracy: 0.9801
- val_loss: 0.0902 - val_accuracy: 0.9815
Epoch 306/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0788 - accuracy: 0.9802
- val_loss: 0.0920 - val_accuracy: 0.9815
Epoch 307/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0743 - accuracy: 0.9816
- val_loss: 0.0931 - val_accuracy: 0.9815
Epoch 308/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0794 - accuracy: 0.9789
```

- val_loss: 0.0884 - val_accuracy: 0.9814
Epoch 309/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0862 - accuracy: 0.9771
- val_loss: 0.0924 - val_accuracy: 0.9813
Epoch 310/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0753 - accuracy: 0.9801
- val_loss: 0.0918 - val_accuracy: 0.9815
Epoch 311/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0700 - accuracy: 0.9827
- val_loss: 0.0904 - val_accuracy: 0.9814
Epoch 312/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0752 - accuracy: 0.9800
- val_loss: 0.0897 - val_accuracy: 0.9814
Epoch 313/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0839 - accuracy: 0.9772
- val_loss: 0.0887 - val_accuracy: 0.9815
Epoch 314/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0748 - accuracy: 0.9808
- val_loss: 0.0947 - val_accuracy: 0.9815
Epoch 315/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0772 - accuracy: 0.9798
- val_loss: 0.0901 - val_accuracy: 0.9811
Epoch 316/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0801 - accuracy: 0.9790
- val_loss: 0.0898 - val_accuracy: 0.9810
Epoch 317/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0781 - accuracy: 0.9794
- val_loss: 0.0990 - val_accuracy: 0.9797
Epoch 318/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0721 - accuracy: 0.9817
- val_loss: 0.0931 - val_accuracy: 0.9810
Epoch 319/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0754 - accuracy: 0.9801
- val_loss: 0.0913 - val_accuracy: 0.9813
Epoch 320/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0764 - accuracy: 0.9784
- val_loss: 0.0965 - val_accuracy: 0.9815
Epoch 321/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0706 - accuracy: 0.9817
- val_loss: 0.0949 - val_accuracy: 0.9814
Epoch 322/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0816 - accuracy: 0.9777
- val_loss: 0.0946 - val_accuracy: 0.9815
Epoch 323/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0748 - accuracy: 0.9798
- val_loss: 0.0917 - val_accuracy: 0.9814
Epoch 324/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0689 - accuracy: 0.9820
- val_loss: 0.1034 - val_accuracy: 0.9812
Epoch 325/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0903 - accuracy: 0.9779
- val_loss: 0.0970 - val_accuracy: 0.9810
Epoch 326/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0682 - accuracy: 0.9823
- val_loss: 0.0943 - val_accuracy: 0.9815
Epoch 327/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0677 - accuracy: 0.9814
- val_loss: 0.0942 - val_accuracy: 0.9811
Epoch 328/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0734 - accuracy: 0.9809
- val_loss: 0.0921 - val_accuracy: 0.9815
Epoch 329/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0695 - accuracy: 0.9816
- val_loss: 0.0902 - val_accuracy: 0.9814
Epoch 330/600

273/273 [==============================]- 1s 4ms/step - loss: 0.0788 - accuracy: 0.9791 - val_loss: 0.0906 - val_accuracy: 0.9812
Epoch 331/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0674 - accuracy: 0.9825 - val_loss: 0.0924 - val_accuracy: 0.9814
Epoch 332/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0727 - accuracy: 0.9811 - val_loss: 0.0922 - val_accuracy: 0.9810
Epoch 333/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0705 - accuracy: 0.9818 - val_loss: 0.0983 - val_accuracy: 0.9812
Epoch 334/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0730 - accuracy: 0.9802 - val_loss: 0.0918 - val_accuracy: 0.9812
Epoch 335/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0693 - accuracy: 0.9819 - val_loss: 0.0916 - val_accuracy: 0.9811
Epoch 336/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0771 - accuracy: 0.9785 - val_loss: 0.0910 - val_accuracy: 0.9811
Epoch 337/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0711 - accuracy: 0.9816 - val_loss: 0.0914 - val_accuracy: 0.9810
Epoch 338/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0756 - accuracy: 0.9801 - val_loss: 0.0933 - val_accuracy: 0.9814
Epoch 339/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0769 - accuracy: 0.9789 - val_loss: 0.0915 - val_accuracy: 0.9811
Epoch 340/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0786 - accuracy: 0.9789 - val_loss: 0.0925 - val_accuracy: 0.9810
Epoch 341/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0783 - accuracy: 0.9785 - val_loss: 0.0919 - val_accuracy: 0.9814
Epoch 342/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0626 - accuracy: 0.9837 - val_loss: 0.0927 - val_accuracy: 0.9807
Epoch 343/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0705 - accuracy: 0.9808 - val_loss: 0.0952 - val_accuracy: 0.9811
Epoch 344/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0721 - accuracy: 0.9801 - val_loss: 0.0911 - val_accuracy: 0.9815
Epoch 345/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0748 - accuracy: 0.9805 - val_loss: 0.0988 - val_accuracy: 0.9815
Epoch 346/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0746 - accuracy: 0.9810 - val_loss: 0.1001 - val_accuracy: 0.9815
Epoch 347/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0754 - accuracy: 0.9797 - val_loss: 0.0972 - val_accuracy: 0.9814
Epoch 348/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0686 - accuracy: 0.9819 - val_loss: 0.0940 - val_accuracy: 0.9812
Epoch 349/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0761 - accuracy: 0.9800 - val_loss: 0.0975 - val_accuracy: 0.9815
Epoch 350/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0692 - accuracy: 0.9828 - val_loss: 0.0972 - val_accuracy: 0.9804
Epoch 351/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0799 - accuracy: 0.9787 - val_loss: 0.0923 - val_accuracy: 0.9813

Epoch 352/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0731 - accuracy: 0.9806
- val_loss: 0.0950 - val_accuracy: 0.9809
Epoch 353/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0731 - accuracy: 0.9814
- val_loss: 0.0914 - val_accuracy: 0.9815
Epoch 354/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0686 - accuracy: 0.9820
- val_loss: 0.0933 - val_accuracy: 0.9815
Epoch 355/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0787 - accuracy: 0.9790
- val_loss: 0.0955 - val_accuracy: 0.9814
Epoch 356/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0715 - accuracy: 0.9809
- val_loss: 0.0982 - val_accuracy: 0.9799
Epoch 357/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0765 - accuracy: 0.9801
- val_loss: 0.0978 - val_accuracy: 0.9808
Epoch 358/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0730 - accuracy: 0.9803
- val_loss: 0.0946 - val_accuracy: 0.9813
Epoch 359/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0712 - accuracy: 0.9804
- val_loss: 0.0943 - val_accuracy: 0.9798
Epoch 360/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0716 - accuracy: 0.9810
- val_loss: 0.0973 - val_accuracy: 0.9810
Epoch 361/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0645 - accuracy: 0.9837
- val_loss: 0.1040 - val_accuracy: 0.9810
Epoch 362/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0731 - accuracy: 0.9813
- val_loss: 0.0942 - val_accuracy: 0.9808
Epoch 363/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0717 - accuracy: 0.9817
- val_loss: 0.0953 - val_accuracy: 0.9812
Epoch 364/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0758 - accuracy: 0.9783
- val_loss: 0.0986 - val_accuracy: 0.9809
Epoch 365/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0602 - accuracy: 0.9848
- val_loss: 0.1022 - val_accuracy: 0.9807
Epoch 366/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0687 - accuracy: 0.9826
- val_loss: 0.0973 - val_accuracy: 0.9801
Epoch 367/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0747 - accuracy: 0.9816
- val_loss: 0.0952 - val_accuracy: 0.9811
Epoch 368/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0724 - accuracy: 0.9811
- val_loss: 0.0951 - val_accuracy: 0.9811
Epoch 369/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0746 - accuracy: 0.9803
- val_loss: 0.0899 - val_accuracy: 0.9813
Epoch 370/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0714 - accuracy: 0.9805
- val_loss: 0.0959 - val_accuracy: 0.9810
Epoch 371/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0775 - accuracy: 0.9785
- val_loss: 0.1006 - val_accuracy: 0.9814
Epoch 372/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0701 - accuracy: 0.9829
- val_loss: 0.0970 - val_accuracy: 0.9813
Epoch 373/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0740 - accuracy: 0.9793

- val_loss: 0.0970 - val_accuracy: 0.9815
Epoch 374/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0731 - accuracy: 0.9802 - val_loss: 0.1010 - val_accuracy: 0.9812
Epoch 375/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0734 - accuracy: 0.9814 - val_loss: 0.0944 - val_accuracy: 0.9815
Epoch 376/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0819 - accuracy: 0.9793 - val_loss: 0.0974 - val_accuracy: 0.9815
Epoch 377/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0684 - accuracy: 0.9819 - val_loss: 0.0998 - val_accuracy: 0.9809
Epoch 378/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0796 - accuracy: 0.9795 - val_loss: 0.1066 - val_accuracy: 0.9812
Epoch 379/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0827 - accuracy: 0.9790 - val_loss: 0.0960 - val_accuracy: 0.9806
Epoch 380/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0723 - accuracy: 0.9809 - val_loss: 0.0975 - val_accuracy: 0.9811
Epoch 381/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0733 - accuracy: 0.9817 - val_loss: 0.0938 - val_accuracy: 0.9804
Epoch 382/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0713 - accuracy: 0.9815 - val_loss: 0.0948 - val_accuracy: 0.9810
Epoch 383/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0743 - accuracy: 0.9797 - val_loss: 0.0954 - val_accuracy: 0.9808
Epoch 384/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0712 - accuracy: 0.9819 - val_loss: 0.0971 - val_accuracy: 0.9790
Epoch 385/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0706 - accuracy: 0.9808 - val_loss: 0.0964 - val_accuracy: 0.9809
Epoch 386/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0663 - accuracy: 0.9820 - val_loss: 0.0951 - val_accuracy: 0.9811
Epoch 387/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0728 - accuracy: 0.9798 - val_loss: 0.0935 - val_accuracy: 0.9815
Epoch 388/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0644 - accuracy: 0.9839 - val_loss: 0.0980 - val_accuracy: 0.9810
Epoch 389/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0742 - accuracy: 0.9811 - val_loss: 0.0956 - val_accuracy: 0.9814
Epoch 390/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0833 - accuracy: 0.9758 - val_loss: 0.0948 - val_accuracy: 0.9803
Epoch 391/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0722 - accuracy: 0.9818 - val_loss: 0.0991 - val_accuracy: 0.9808
Epoch 392/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0681 - accuracy: 0.9820 - val_loss: 0.0959 - val_accuracy: 0.9803
Epoch 393/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0781 - accuracy: 0.9797 - val_loss: 0.0983 - val_accuracy: 0.9810
Epoch 394/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0736 - accuracy: 0.9801 - val_loss: 0.0985 - val_accuracy: 0.9814
Epoch 395/600

```
273/273 [==============================]- 1s 4ms/step - loss: 0.0622 - accuracy: 0.9846
- val_loss: 0.0977 - val_accuracy: 0.9810
Epoch 396/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0712 - accuracy: 0.9816
- val_loss: 0.0964 - val_accuracy: 0.9807
Epoch 397/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0644 - accuracy: 0.9816
- val_loss: 0.0893 - val_accuracy: 0.9807
Epoch 398/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0774 - accuracy: 0.9803
- val_loss: 0.0896 - val_accuracy: 0.9813
Epoch 399/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0755 - accuracy: 0.9805
- val_loss: 0.0912 - val_accuracy: 0.9810
Epoch 400/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0750 - accuracy: 0.9801
- val_loss: 0.0963 - val_accuracy: 0.9804
Epoch 401/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0733 - accuracy: 0.9812
- val_loss: 0.0930 - val_accuracy: 0.9809
Epoch 402/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0773 - accuracy: 0.9796
- val_loss: 0.0989 - val_accuracy: 0.9782
Epoch 403/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0749 - accuracy: 0.9787
- val_loss: 0.0940 - val_accuracy: 0.9804
Epoch 404/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0735 - accuracy: 0.9802
- val_loss: 0.0961 - val_accuracy: 0.9802
Epoch 405/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0671 - accuracy: 0.9820
- val_loss: 0.0911 - val_accuracy: 0.9805
Epoch 406/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0766 - accuracy: 0.9789
- val_loss: 0.0947 - val_accuracy: 0.9804
Epoch 407/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0730 - accuracy: 0.9808
- val_loss: 0.0933 - val_accuracy: 0.9814
Epoch 408/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0771 - accuracy: 0.9799
- val_loss: 0.0953 - val_accuracy: 0.9806
Epoch 409/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0798 - accuracy: 0.9789
- val_loss: 0.0993 - val_accuracy: 0.9803
Epoch 410/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0642 - accuracy: 0.9830
- val_loss: 0.0980 - val_accuracy: 0.9809
Epoch 411/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0763 - accuracy: 0.9781
- val_loss: 0.0974 - val_accuracy: 0.9803
Epoch 412/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0697 - accuracy: 0.9811
- val_loss: 0.0969 - val_accuracy: 0.9812
Epoch 413/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0724 - accuracy: 0.9794
- val_loss: 0.0997 - val_accuracy: 0.9809
Epoch 414/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0723 - accuracy: 0.9807
- val_loss: 0.1090 - val_accuracy: 0.9805
Epoch 415/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0809 - accuracy: 0.9797
- val_loss: 0.0961 - val_accuracy: 0.9797
Epoch 416/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0703 - accuracy: 0.9817
- val_loss: 0.0994 - val_accuracy: 0.9806
```

Epoch 417/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0835 - accuracy: 0.9758 - val_loss: 0.1002 - val_accuracy: 0.9806
Epoch 418/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0626 - accuracy: 0.9833 - val_loss: 0.1003 - val_accuracy: 0.9809
Epoch 419/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0716 - accuracy: 0.9805 - val_loss: 0.0985 - val_accuracy: 0.9803
Epoch 420/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0756 - accuracy: 0.9797 - val_loss: 0.1030 - val_accuracy: 0.9810
Epoch 421/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0631 - accuracy: 0.9841 - val_loss: 0.0993 - val_accuracy: 0.9809
Epoch 422/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0786 - accuracy: 0.9785 - val_loss: 0.1039 - val_accuracy: 0.9809
Epoch 423/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0764 - accuracy: 0.9801 - val_loss: 0.1002 - val_accuracy: 0.9808
Epoch 424/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0694 - accuracy: 0.9807 - val_loss: 0.1070 - val_accuracy: 0.9795
Epoch 425/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0690 - accuracy: 0.9814 - val_loss: 0.1197 - val_accuracy: 0.9811
Epoch 426/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0805 - accuracy: 0.9794 - val_loss: 0.1016 - val_accuracy: 0.9811
Epoch 427/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0701 - accuracy: 0.9821 - val_loss: 0.1006 - val_accuracy: 0.9812
Epoch 428/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0792 - accuracy: 0.9784 - val_loss: 0.1060 - val_accuracy: 0.9809
Epoch 429/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0695 - accuracy: 0.9807 - val_loss: 0.1035 - val_accuracy: 0.9809
Epoch 430/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0741 - accuracy: 0.9792 - val_loss: 0.1118 - val_accuracy: 0.9802
Epoch 431/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0770 - accuracy: 0.9786 - val_loss: 0.1050 - val_accuracy: 0.9808
Epoch 432/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0700 - accuracy: 0.9824 - val_loss: 0.1004 - val_accuracy: 0.9792
Epoch 433/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0807 - accuracy: 0.9791 - val_loss: 0.0981 - val_accuracy: 0.9802
Epoch 434/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0622 - accuracy: 0.9835 - val_loss: 0.0992 - val_accuracy: 0.9809
Epoch 435/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0674 - accuracy: 0.9821 - val_loss: 0.1056 - val_accuracy: 0.9805
Epoch 436/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0608 - accuracy: 0.9831 - val_loss: 0.1072 - val_accuracy: 0.9797
Epoch 437/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0712 - accuracy: 0.9806 - val_loss: 0.1089 - val_accuracy: 0.9803
Epoch 438/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0728 - accuracy: 0.9803

```
– val_loss: 0.0960 - val_accuracy: 0.9809
Epoch 439/600
273/273 [==============================] - 1s 3ms/step - loss: 0.0660 - accuracy: 0.9833
– val_loss: 0.1032 - val_accuracy: 0.9809
Epoch 440/600
273/273 [==============================] - 1s 4ms/step - loss: 0.0689 - accuracy: 0.9804
– val_loss: 0.1037 - val_accuracy: 0.9814
Epoch 441/600
273/273 [==============================] - 1s 4ms/step - loss: 0.0793 - accuracy: 0.9784
– val_loss: 0.1005 - val_accuracy: 0.9808
Epoch 442/600
273/273 [==============================] - 1s 4ms/step - loss: 0.0730 - accuracy: 0.9804
– val_loss: 0.1019 - val_accuracy: 0.9812
Epoch 443/600
273/273 [==============================] - 1s 3ms/step - loss: 0.0666 - accuracy: 0.9835
– val_loss: 0.1025 - val_accuracy: 0.9812
Epoch 444/600
273/273 [==============================] - 1s 3ms/step - loss: 0.0691 - accuracy: 0.9816
– val_loss: 0.0986 - val_accuracy: 0.9808
Epoch 445/600
273/273 [==============================] - 1s 4ms/step - loss: 0.0757 - accuracy: 0.9807
– val_loss: 0.1010 - val_accuracy: 0.9792
Epoch 446/600
273/273 [==============================] - 1s 4ms/step - loss: 0.0688 - accuracy: 0.9815
– val_loss: 0.1062 - val_accuracy: 0.9813
Epoch 447/600
273/273 [==============================] - 1s 4ms/step - loss: 0.0710 - accuracy: 0.9804
– val_loss: 0.1039 - val_accuracy: 0.9797
Epoch 448/600
273/273 [==============================] - 1s 4ms/step - loss: 0.0674 - accuracy: 0.9824
– val_loss: 0.1054 - val_accuracy: 0.9808
Epoch 449/600
273/273 [==============================] - 1s 4ms/step - loss: 0.0673 - accuracy: 0.9818
– val_loss: 0.0994 - val_accuracy: 0.9806
Epoch 450/600
273/273 [==============================] - 1s 4ms/step - loss: 0.0636 - accuracy: 0.9824
– val_loss: 0.1074 - val_accuracy: 0.9808
Epoch 451/600
273/273 [==============================] - 1s 4ms/step - loss: 0.0700 - accuracy: 0.9815
– val_loss: 0.0963 - val_accuracy: 0.9810
Epoch 452/600
273/273 [==============================] - 1s 4ms/step - loss: 0.0677 - accuracy: 0.9820
– val_loss: 0.1005 - val_accuracy: 0.9810
Epoch 453/600
273/273 [==============================] - 1s 4ms/step - loss: 0.0723 - accuracy: 0.9793
– val_loss: 0.1000 - val_accuracy: 0.9808
Epoch 454/600
273/273 [==============================] - 1s 4ms/step - loss: 0.0778 - accuracy: 0.9786
– val_loss: 0.1152 - val_accuracy: 0.9812
Epoch 455/600
273/273 [==============================] - 1s 4ms/step - loss: 0.0697 - accuracy: 0.9816
– val_loss: 0.1021 - val_accuracy: 0.9812
Epoch 456/600
273/273 [==============================] - 1s 4ms/step - loss: 0.0764 - accuracy: 0.9794
– val_loss: 0.1080 - val_accuracy: 0.9812
Epoch 457/600
273/273 [==============================] - 1s 4ms/step - loss: 0.0687 - accuracy: 0.9820
– val_loss: 0.1092 - val_accuracy: 0.9800
Epoch 458/600
273/273 [==============================] - 1s 4ms/step - loss: 0.0711 - accuracy: 0.9807
– val_loss: 0.1042 - val_accuracy: 0.9803
Epoch 459/600
273/273 [==============================] - 1s 4ms/step - loss: 0.0770 - accuracy: 0.9790
– val_loss: 0.1018 - val_accuracy: 0.9810
Epoch 460/600
```

273/273 [==============================]- 1s 4ms/step - loss: 0.0629 - accuracy: 0.9840 - val_loss: 0.1060 - val_accuracy: 0.9811
Epoch 461/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0793 - accuracy: 0.9797 - val_loss: 0.1045 - val_accuracy: 0.9803
Epoch 462/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0667 - accuracy: 0.9816 - val_loss: 0.1040 - val_accuracy: 0.9810
Epoch 463/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0676 - accuracy: 0.9822 - val_loss: 0.1093 - val_accuracy: 0.9790
Epoch 464/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0761 - accuracy: 0.9788 - val_loss: 0.0942 - val_accuracy: 0.9804
Epoch 465/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0807 - accuracy: 0.9782 - val_loss: 0.1080 - val_accuracy: 0.9814
Epoch 466/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0705 - accuracy: 0.9795 - val_loss: 0.1173 - val_accuracy: 0.9811
Epoch 467/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0680 - accuracy: 0.9832 - val_loss: 0.1030 - val_accuracy: 0.9800
Epoch 468/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0748 - accuracy: 0.9800 - val_loss: 0.1041 - val_accuracy: 0.9804
Epoch 469/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0711 - accuracy: 0.9824 - val_loss: 0.1040 - val_accuracy: 0.9809
Epoch 470/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0709 - accuracy: 0.9810 - val_loss: 0.1064 - val_accuracy: 0.9790
Epoch 471/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0770 - accuracy: 0.9788 - val_loss: 0.1038 - val_accuracy: 0.9809
Epoch 472/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0659 - accuracy: 0.9826 - val_loss: 0.1079 - val_accuracy: 0.9809
Epoch 473/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0666 - accuracy: 0.9825 - val_loss: 0.0991 - val_accuracy: 0.9807
Epoch 474/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0756 - accuracy: 0.9780 - val_loss: 0.1046 - val_accuracy: 0.9800
Epoch 475/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0670 - accuracy: 0.9826 - val_loss: 0.1064 - val_accuracy: 0.9804
Epoch 476/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0686 - accuracy: 0.9820 - val_loss: 0.1059 - val_accuracy: 0.9812
Epoch 477/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0729 - accuracy: 0.9811 - val_loss: 0.1066 - val_accuracy: 0.9809
Epoch 478/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0700 - accuracy: 0.9819 - val_loss: 0.1068 - val_accuracy: 0.9805
Epoch 479/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0693 - accuracy: 0.9813 - val_loss: 0.1035 - val_accuracy: 0.9810
Epoch 480/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0777 - accuracy: 0.9791 - val_loss: 0.1170 - val_accuracy: 0.9798
Epoch 481/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0691 - accuracy: 0.9806 - val_loss: 0.1105 - val_accuracy: 0.9811

```
Epoch 482/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0664 - accuracy: 0.9825
- val_loss: 0.1055 - val_accuracy: 0.9806
Epoch 483/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0650 - accuracy: 0.9825
- val_loss: 0.1038 - val_accuracy: 0.9803
Epoch 484/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0676 - accuracy: 0.9815
- val_loss: 0.1056 - val_accuracy: 0.9809
Epoch 485/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0716 - accuracy: 0.9807
- val_loss: 0.1102 - val_accuracy: 0.9810
Epoch 486/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0749 - accuracy: 0.9802
- val_loss: 0.1015 - val_accuracy: 0.9805
Epoch 487/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0670 - accuracy: 0.9829
- val_loss: 0.1104 - val_accuracy: 0.9810
Epoch 488/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0718 - accuracy: 0.9811
- val_loss: 0.1084 - val_accuracy: 0.9812
Epoch 489/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0695 - accuracy: 0.9805
- val_loss: 0.1093 - val_accuracy: 0.9806
Epoch 490/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0686 - accuracy: 0.9821
- val_loss: 0.1007 - val_accuracy: 0.9799
Epoch 491/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0684 - accuracy: 0.9827
- val_loss: 0.1109 - val_accuracy: 0.9806
Epoch 492/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0707 - accuracy: 0.9828
- val_loss: 0.1040 - val_accuracy: 0.9812
Epoch 493/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0721 - accuracy: 0.9794
- val_loss: 0.1125 - val_accuracy: 0.9807
Epoch 494/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0671 - accuracy: 0.9825
- val_loss: 0.0978 - val_accuracy: 0.9808
Epoch 495/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0772 - accuracy: 0.9798
- val_loss: 0.1043 - val_accuracy: 0.9800
Epoch 496/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0629 - accuracy: 0.9837
- val_loss: 0.1043 - val_accuracy: 0.9806
Epoch 497/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0676 - accuracy: 0.9821
- val_loss: 0.1034 - val_accuracy: 0.9803
Epoch 498/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0667 - accuracy: 0.9809
- val_loss: 0.1047 - val_accuracy: 0.9809
Epoch 499/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0667 - accuracy: 0.9836
- val_loss: 0.1074 - val_accuracy: 0.9802
Epoch 500/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0780 - accuracy: 0.9790
- val_loss: 0.1130 - val_accuracy: 0.9814
Epoch 501/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0698 - accuracy: 0.9816
- val_loss: 0.1017 - val_accuracy: 0.9801
Epoch 502/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0671 - accuracy: 0.9814
- val_loss: 0.1013 - val_accuracy: 0.9805
Epoch 503/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0651 - accuracy: 0.9837
```

- val_loss: 0.1064 - val_accuracy: 0.9800
Epoch 504/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0690 - accuracy: 0.9809 - val_loss: 0.1031 - val_accuracy: 0.9799
Epoch 505/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0705 - accuracy: 0.9813 - val_loss: 0.1021 - val_accuracy: 0.9800
Epoch 506/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0685 - accuracy: 0.9812 - val_loss: 0.1073 - val_accuracy: 0.9812
Epoch 507/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0685 - accuracy: 0.9807 - val_loss: 0.1052 - val_accuracy: 0.9802
Epoch 508/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0712 - accuracy: 0.9809 - val_loss: 0.1039 - val_accuracy: 0.9810
Epoch 509/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0628 - accuracy: 0.9839 - val_loss: 0.1014 - val_accuracy: 0.9813
Epoch 510/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0765 - accuracy: 0.9773 - val_loss: 0.1045 - val_accuracy: 0.9810
Epoch 511/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0760 - accuracy: 0.9794 - val_loss: 0.1072 - val_accuracy: 0.9805
Epoch 512/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0675 - accuracy: 0.9827 - val_loss: 0.1244 - val_accuracy: 0.9802
Epoch 513/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0770 - accuracy: 0.9795 - val_loss: 0.1024 - val_accuracy: 0.9800
Epoch 514/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0623 - accuracy: 0.9842 - val_loss: 0.1024 - val_accuracy: 0.9809
Epoch 515/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0685 - accuracy: 0.9816 - val_loss: 0.1113 - val_accuracy: 0.9811
Epoch 516/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0622 - accuracy: 0.9836 - val_loss: 0.1056 - val_accuracy: 0.9811
Epoch 517/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0693 - accuracy: 0.9812 - val_loss: 0.0974 - val_accuracy: 0.9806
Epoch 518/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0690 - accuracy: 0.9818 - val_loss: 0.1061 - val_accuracy: 0.9798
Epoch 519/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0741 - accuracy: 0.9804 - val_loss: 0.1020 - val_accuracy: 0.9803
Epoch 520/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0715 - accuracy: 0.9804 - val_loss: 0.1063 - val_accuracy: 0.9805
Epoch 521/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0592 - accuracy: 0.9847 - val_loss: 0.1024 - val_accuracy: 0.9807
Epoch 522/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0713 - accuracy: 0.9809 - val_loss: 0.1073 - val_accuracy: 0.9802
Epoch 523/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0674 - accuracy: 0.9819 - val_loss: 0.1052 - val_accuracy: 0.9807
Epoch 524/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0708 - accuracy: 0.9809 - val_loss: 0.1117 - val_accuracy: 0.9804
Epoch 525/600

273/273 [==============================]- 1s 4ms/step - loss: 0.0804 - accuracy: 0.9780 - val_loss: 0.1063 - val_accuracy: 0.9798
Epoch 526/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0715 - accuracy: 0.9811 - val_loss: 0.1150 - val_accuracy: 0.9807
Epoch 527/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0773 - accuracy: 0.9791 - val_loss: 0.1016 - val_accuracy: 0.9811
Epoch 528/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0802 - accuracy: 0.9773 - val_loss: 0.1024 - val_accuracy: 0.9807
Epoch 529/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0722 - accuracy: 0.9799 - val_loss: 0.1137 - val_accuracy: 0.9804
Epoch 530/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0697 - accuracy: 0.9812 - val_loss: 0.1122 - val_accuracy: 0.9810
Epoch 531/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0663 - accuracy: 0.9811 - val_loss: 0.1114 - val_accuracy: 0.9804
Epoch 532/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0705 - accuracy: 0.9798 - val_loss: 0.1137 - val_accuracy: 0.9806
Epoch 533/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0639 - accuracy: 0.9827 - val_loss: 0.1173 - val_accuracy: 0.9771
Epoch 534/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0734 - accuracy: 0.9796 - val_loss: 0.1176 - val_accuracy: 0.9802
Epoch 535/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0722 - accuracy: 0.9819 - val_loss: 0.1084 - val_accuracy: 0.9805
Epoch 536/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0637 - accuracy: 0.9832 - val_loss: 0.1131 - val_accuracy: 0.9803
Epoch 537/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0797 - accuracy: 0.9773 - val_loss: 0.1131 - val_accuracy: 0.9803
Epoch 538/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0704 - accuracy: 0.9812 - val_loss: 0.1082 - val_accuracy: 0.9799
Epoch 539/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0730 - accuracy: 0.9805 - val_loss: 0.1057 - val_accuracy: 0.9803
Epoch 540/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0722 - accuracy: 0.9809 - val_loss: 0.1093 - val_accuracy: 0.9801
Epoch 541/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0669 - accuracy: 0.9817 - val_loss: 0.1106 - val_accuracy: 0.9798
Epoch 542/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0692 - accuracy: 0.9808 - val_loss: 0.1074 - val_accuracy: 0.9803
Epoch 543/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0720 - accuracy: 0.9797 - val_loss: 0.1136 - val_accuracy: 0.9800
Epoch 544/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0731 - accuracy: 0.9798 - val_loss: 0.1079 - val_accuracy: 0.9807
Epoch 545/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0656 - accuracy: 0.9822 - val_loss: 0.1130 - val_accuracy: 0.9803
Epoch 546/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0663 - accuracy: 0.9828 - val_loss: 0.1021 - val_accuracy: 0.9804

Epoch 547/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0649 - accuracy: 0.9827 - val_loss: 0.1029 - val_accuracy: 0.9805
Epoch 548/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0664 - accuracy: 0.9820 - val_loss: 0.1080 - val_accuracy: 0.9805
Epoch 549/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0696 - accuracy: 0.9805 - val_loss: 0.1054 - val_accuracy: 0.9803
Epoch 550/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0692 - accuracy: 0.9814 - val_loss: 0.1093 - val_accuracy: 0.9800
Epoch 551/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0640 - accuracy: 0.9832 - val_loss: 0.1084 - val_accuracy: 0.9808
Epoch 552/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0659 - accuracy: 0.9822 - val_loss: 0.1084 - val_accuracy: 0.9800
Epoch 553/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0790 - accuracy: 0.9787 - val_loss: 0.1053 - val_accuracy: 0.9812
Epoch 554/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0693 - accuracy: 0.9808 - val_loss: 0.1048 - val_accuracy: 0.9806
Epoch 555/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0744 - accuracy: 0.9794 - val_loss: 0.1114 - val_accuracy: 0.9807
Epoch 556/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0671 - accuracy: 0.9816 - val_loss: 0.1088 - val_accuracy: 0.9801
Epoch 557/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0685 - accuracy: 0.9823 - val_loss: 0.1113 - val_accuracy: 0.9805
Epoch 558/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0674 - accuracy: 0.9823 - val_loss: 0.1123 - val_accuracy: 0.9805
Epoch 559/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0738 - accuracy: 0.9791 - val_loss: 0.1051 - val_accuracy: 0.9812
Epoch 560/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0681 - accuracy: 0.9817 - val_loss: 0.1072 - val_accuracy: 0.9807
Epoch 561/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0678 - accuracy: 0.9807 - val_loss: 0.1026 - val_accuracy: 0.9807
Epoch 562/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0692 - accuracy: 0.9823 - val_loss: 0.1126 - val_accuracy: 0.9808
Epoch 563/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0595 - accuracy: 0.9842 - val_loss: 0.1109 - val_accuracy: 0.9809
Epoch 564/600
273/273 [==============================]- 1s 3ms/step - loss: 0.0692 - accuracy: 0.9802 - val_loss: 0.1080 - val_accuracy: 0.9812
Epoch 565/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0749 - accuracy: 0.9796 - val_loss: 0.1162 - val_accuracy: 0.9801
Epoch 566/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0633 - accuracy: 0.9819 - val_loss: 0.1212 - val_accuracy: 0.9799
Epoch 567/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0740 - accuracy: 0.9796 - val_loss: 0.1201 - val_accuracy: 0.9808
Epoch 568/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0665 - accuracy: 0.9825

```
- val_loss: 0.1113 - val_accuracy: 0.9801
Epoch 569/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0704 - accuracy: 0.9801
- val_loss: 0.1077 - val_accuracy: 0.9809
Epoch 570/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0703 - accuracy: 0.9813
- val_loss: 0.1112 - val_accuracy: 0.9803
Epoch 571/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0612 - accuracy: 0.9838
- val_loss: 0.1112 - val_accuracy: 0.9803
Epoch 572/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0741 - accuracy: 0.9804
- val_loss: 0.1125 - val_accuracy: 0.9805
Epoch 573/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0647 - accuracy: 0.9830
- val_loss: 0.1223 - val_accuracy: 0.9804
Epoch 574/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0671 - accuracy: 0.9820
- val_loss: 0.1176 - val_accuracy: 0.9802
Epoch 575/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0611 - accuracy: 0.9838
- val_loss: 0.1206 - val_accuracy: 0.9806
Epoch 576/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0727 - accuracy: 0.9808
- val_loss: 0.1207 - val_accuracy: 0.9808
Epoch 577/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0674 - accuracy: 0.9820
- val_loss: 0.1159 - val_accuracy: 0.9808
Epoch 578/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0610 - accuracy: 0.9830
- val_loss: 0.1182 - val_accuracy: 0.9800
Epoch 579/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0686 - accuracy: 0.9822
- val_loss: 0.1196 - val_accuracy: 0.9772
Epoch 580/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0665 - accuracy: 0.9820
- val_loss: 0.1120 - val_accuracy: 0.9792
Epoch 581/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0748 - accuracy: 0.9799
- val_loss: 0.1069 - val_accuracy: 0.9806
Epoch 582/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0793 - accuracy: 0.9794
- val_loss: 0.1088 - val_accuracy: 0.9805
Epoch 583/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0701 - accuracy: 0.9810
- val_loss: 0.1109 - val_accuracy: 0.9799
Epoch 584/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0679 - accuracy: 0.9819
- val_loss: 0.1114 - val_accuracy: 0.9810
Epoch 585/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0573 - accuracy: 0.9842
- val_loss: 0.1099 - val_accuracy: 0.9806
Epoch 586/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0691 - accuracy: 0.9814
- val_loss: 0.1076 - val_accuracy: 0.9810
Epoch 587/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0713 - accuracy: 0.9802
- val_loss: 0.1209 - val_accuracy: 0.9808
Epoch 588/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0641 - accuracy: 0.9833
- val_loss: 0.1175 - val_accuracy: 0.9791
Epoch 589/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0641 - accuracy: 0.9823
- val_loss: 0.1180 - val_accuracy: 0.9808
Epoch 590/600
```

273/273 [==============================]- 1s 3ms/step - loss: 0.0677 - accuracy: 0.9823 - val_loss: 0.1176 - val_accuracy: 0.9791
Epoch 591/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0592 - accuracy: 0.9846 - val_loss: 0.1309 - val_accuracy: 0.9802
Epoch 592/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0598 - accuracy: 0.9839 - val_loss: 0.0995 - val_accuracy: 0.9797
Epoch 593/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0708 - accuracy: 0.9799 - val_loss: 0.1063 - val_accuracy: 0.9800
Epoch 594/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0692 - accuracy: 0.9820 - val_loss: 0.1166 - val_accuracy: 0.9808
Epoch 595/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0597 - accuracy: 0.9842 - val_loss: 0.1175 - val_accuracy: 0.9804
Epoch 596/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0640 - accuracy: 0.9820 - val_loss: 0.1158 - val_accuracy: 0.9805
Epoch 597/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0713 - accuracy: 0.9803 - val_loss: 0.1141 - val_accuracy: 0.9805
Epoch 598/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0632 - accuracy: 0.9836 - val_loss: 0.1136 - val_accuracy: 0.9803
Epoch 599/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0638 - accuracy: 0.9825 - val_loss: 0.1322 - val_accuracy: 0.9806
Epoch 600/600
273/273 [==============================]- 1s 4ms/step - loss: 0.0669 - accuracy: 0.9827 - val_loss: 0.1083 - val_accuracy: 0.9808

Out[85]: <tensorflow.python.keras.callbacks.History at 0x7f721f276750>

In [94]:
```
predictions =ann_model.predict(x_test)[:,0]
```

Out[94]:
```
array([5.1400346e-01, 6.3963707e-10, 2.3623914e-02,
       8.3117070e-29, 5.2477092e-02, 9.6591539e-06,
       9.7921491e-04, 2.3538500e-02,
       8.6560249e-03, 6.8458521e-06], dtype=float32)
```

In [100…
```
predictions = np.round(ann_model.predict(x_test)[:,0
```

Out[100…
```
array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
dtype=float32)
```

CONFUSION MATRIX AND CLASIFICATION REPORT TO EVALUATE PERFORMANCE

In [89]:
```
from sklearn.metrics import classification_report ,
```

In [102…
```
print(classification_report(y_test,predictions))
```

	precision	recall	f1-score	support
0	0.98	1.00	0.99	19970
1	0.11	0.01	0.01	375

accuracy			0.98	20345
macro avg	0.54	0.50	0.50	20345
weighted avg	0.97	0.98	0.97	20345

In [103…

```
print(confusion_matrix(y_test,predictions))
```

```
[[19953    17]
 [  373     2]]
```

In [ ]:

CONCLUSION:

Stroke is the leading cause of adult disability worldwide, with up to two-thirds of individuals experiencing long-term disabilities. Large-scale neuroimaging studies have shown promise in identifying robust biomarkers (e.g., measures of brain structure) of long-term stroke recovery following rehabilitation. However, analyzing large rehabilitation-related datasets is problematic due to barriers in accurate stroke lesion segmentation. Manually-traced lesions are currently the gold standard for lesion segmentation on T1-weighted MRIs, but are labor intensive and require anatomical expertise. While algorithms have been developed to automate this process, the results often lack accuracy. Newer algorithms that employ machine-learning techniques are promising, yet these require large training datasets to optimize performance. This large, diverse dataset can be used to train and test lesion segmentation algorithms and provides a standardized dataset for comparing the performance of different segmentation methods.

Approximately 795,000 people in the United States suffer from a stroke every year, resulting in nearly 133,000 deaths. In addition, up to 2/3 of stroke survivors experience long-term disabilities that impair their participation in daily activities. Careful clinical decision making is thus critical both at the acute stage, where interventions can spare neural tissue or be used to promote early functional recovery, and at the subacute/chronic stages, where effective rehabilitation can promote long-term functional recovery. Enormous efforts have been made to predict outcomes and response to treatments at both acute and subacute/chronic stages using brain imaging.

This dataset is used to predict whether a patient is likely to get stroke based on the input parameters like gender, age, and various diseases and smoking status. A subset of the original train data is taken using the filtering method for Machine Learning and Data Visualization.

This project aims to identify the risk factors for stroke. The patient data was obtained from the drive sent by our instructor. Methods to ascertain whether a variable is a risk factor were described. Results were visualized and discovered insights were discussed. It is ended with a conclusion and some ideas were suggested for future work purposes.

Bibliography:

The contents have been gathered from the following:

1. **Information:**

- https://towardsdatascience.com/
- https://data-flair.training/blogs/artificial-neural-networks-for-machine-learning/
- https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/
- https://www.python.org/
- https://www.google.com/search?q=color+picker
- https://www.wikipedia.com/
- https://www.geeksforgeeks.com/

2. **Code and Snapshots: Self-performed**