ABSTRACT

Our study introduces a novel approach, combining deep neural networks with transfer learning, to predict stroke risk. Utilizing a healthcare dataset, we preprocess the data, encode categorical variables, and train Decision Tree and Random Forest classifiers. Results highlight the hybrid model's effectiveness in accurately predicting stroke risk, showcasing its potential to augment healthcare analytics and provide valuable insights for preventive interventions. This hybrid deep transfer learning framework offers a promising avenue for enhancing stroke risk prediction models, thereby contributing to improved patient care and health results.

CHAPTER 1

INTRODUCTION

1.1 General Introduction:

Stroke is a major worldwide health issue which causes serious challenges with regard to the rates of death and disability. Stroke is the most common cause of long-term disability, which involves a sudden cessation of blood flow to the brain that causes serious neurological deficits. Even with advances in medical research, minimizing the negative effects of stroke still depends on early detection and treatment. Stroke risk assessment has always been based on known variables including age, diabetes, and hypertension. To improve the precision and accuracy of stroke prediction models, however, new developments in machine learning and healthcare analytics provide encouraging ways forward. Machine learning algorithms are able to reveal complex patterns and associations by analyzing large datasets that include a variety of demographic, lifestyle, and clinical information. This allows us for more accurate prediction of a person's risk of brain stroke so that people can take care of there selfs in earlier stage.

Healthcare analytics can be revolutionized by machine learning algorithms, a branch of artificial intelligence that can identify intricate relationships across vast and diverse datasets. When compared to traditional risk assessment methods alone, these algorithms can predict an individual's likelihood of having a stroke with superior accuracy by analyzing a wide range of factors, including age, gender, heart disease, hypertension, average glucose level, body mass index (BMI), and smoking status, among others. In this study, we use a large and diverse dataset that includes a range of demographic, lifestyle, and healthrelated factors to conduct a thorough investigation of machine learning-based stroke prediction.

Our approach entails the utilization of multiple classification algorithms, including Logistic Regression, Support Vector Classifier (SVC), Decision Tree Classifier (DTC), Random Forest.
Classifier (RFC), Gradient Boosting Classifier (GBC), AdaBoost Classifier (ABC), and K Nearest Neighbors Classifier (KNC). Each of these algorithms offers unique advantages and considerations in terms of model complexity, interpretability, and predictive performance. Through rigorous evaluation and comparison of these models' performance in predicting stroke occurrence, our goal is to identify the most effective approach for early detection and intervention strategies.

Furthermore, our project delves into the assessment of traditional stroke risk factors vis-à-vis machine learning based predictions, providing insights into the added value of incorporating a broader range of variables in stroke prediction. Through interdisciplinary collaboration between machine learning and healthcare domains, we aim to harness the power of data-driven insights to revolutionize stroke prevention and management strategies. By developing reliable and clinically relevant predictive models, we aspire to assist healthcare professionals in identifying individuals at heightened risk of stroke, thereby enabling timely interventions and personalized care plans to reduce the burden of stroke related morbidity and mortality on a population scale. In summary, this project embodies a concerted effort to leverage advanced machine learning techniques and comprehensive datasets to enhance our understanding of stroke risk factors and improve patient outcomes through early detection and intervention strategies. Through interdisciplinary collaboration and rigorous evaluation, we strive to pave the way for more effective stroke prevention and management approaches, ultimately contributing to improved public health outcomes and enhanced quality of life for individuals at risk of stroke.

1.2 Objectives:

The main objective of our project is:

- To predict or to classify the stroke or non-stroke effectively.
- To implement the feature selection for selecting the best features from our dataset.
- To implement the different machine learning algorithm.
- To enhance the overall performance analysis.

CHAPTER 2

SYSTEM PROPOSAL

2.1 EXISTING SYSTEM:

In existing, the system is proposed a novel Hybrid Deep Transfer Learning-based Stroke Risk Prediction (HDTL-SRP) framework which consists of three key components: (1) Generative Instance Transfer (GIT) for making use of the external stroke data distribution among multiple hospitals while preserving the privacy, (2) Network Weight Transfer (NWT) for making use of data from highly correlated diseases (i.e., hypertension or diabetes), (3) Active Instance Transfer (AIT) for balancing the stroke data with the most informative generated instances. It is found that the proposed HDTL-SRP framework outperforms the state-of-the-art SRP models in both synthetic and real-world scenarios.

2.1.1 DISADVANTAGES:

- It is inefficient for huge data volumes.
- Theoretical limits.
- The process is implemented without removing unwanted data.
- The prediction is not accurate.

2.2 PROPOSED SYSTEM:

In this method, the stroke dataset is drawn from the dataset repository. Initially, data preprocessing is performed, which includes addressing missing information in order to avoid erroneous predictions. Proper management of missing values ensures data quality and model fidelity.If there is present any missing values in our input data, we have to replace the missing values by zero or Null values. Then, we have to use label encoding, to encode the label for input data. To encode the columns into numeric values. After that, we can implement the feature selection such as chi square for selecting the best features from pre-processed data. Next, the data must be separated into training and testing sets. This division is critical for evaluating model performance. Once the data has been partitioned, we will use a variety of deep learning and machine learning methods, such as Convolutional Neural Network (CNN), Decision Tree, and Random Forest. Finally, the experimental findings will be evaluated using performance metrics such as accuracy, precision, recall, and the F1-score. These findings can be represented using a comparison graph.

2.2.1 ADVANTAGES:

- . It is beneficial for large datasets.
- To increase the performance metrics results.
- Time consumption is low.
- The process is implemented by removing unwanted data.

LITERATURE REVIEW

2.3.1 Deep Approach for Cholesterol detection and stroke prediction,2023

Authors: Prof. Somashekhar B M, Anusha A N, Namratha N, Shishir S, Sneha N Gangatkar

Methodology:

The proposed methodology for cholesterol and stroke prediction utilizing deep learning entails several steps, beginning with data collection and labelling of eye images to capture diverse blood vessel changes. This approach allows for the utilization of Convolutional Neural Networks (CNNs), specifically FCN and RESNET18 models, renowned for their automatic feature extraction capabilities in image recognition tasks. The CNN architecture involves convolutional layers for feature extraction followed by pooling layers for down sampling, ensuring retention of essential information. However, while CNNs excel in automatic feature extraction, their performance heavily relies on the quality and diversity of the dataset, potentially limiting generalization to unseen data. Preprocessing techniques, such as resizing and normalization, facilitate model training and evaluation. Nonetheless, challenges may arise in handling noise and artifacts present in medical images, impacting model performance. Subsequently, model training involves optimizing the CNNs on the pre-processed dataset, with evaluation conducted using various metrics like accuracy and sensitivity. While this methodology offers promising prospects for automated stroke prediction, its efficacy hinges on robust dataset collection, meticulous preprocessing, and continuous refinement of the CNN architectures. Further exploration of techniques to address dataset biases and enhance model robustness could bolster its applicability in clinical settings, ultimately advancing stroke risk assessment and intervention strategies.

Merits:

- CNNs, such as FCN and RESNET18, enable automatic extraction of relevant features from eye images, eliminating the need for manual feature engineering.
- CNN models can process large volumes of medical images swiftly, enhancing efficiency in clinical practice.

Demerits:

- The performance of CNN models heavily depends on the quality and diversity of the dataset.
- It is bias in algorithmic decision-making, and potential disparities in healthcare delivery.
- CNN models often lack interpretability, making it challenging to understand the underlying rationale behind their predictions, which is crucial for clinical acceptance.

2.3.2 prediction of stoke disease using cnn based approach publish, 2022

Authors: Md. Ashrafuzzaman, Suman Saha, and Kamruddin Nur

Methodology:

By leveraging advanced machine learning techniques, this model can accurately identify patterns and features in medical imaging data associated with stroke. This has led to early detection of stroke risk factors, enabling timely interventions and improving patient outcomes. The CNN's ability to analyse complex images, such as MRI or CT scans, has significantly enhanced the accuracy and reliability of stroke prediction, ultimately contributing to better patient care and reduced healthcare costs. CNN and multilayer perception. The prediction model uses a healthcare data set with eleven features and a single target variable that represents the outcome.

Merits:

- A deep Convolutional Neural Network (CNN) based approach for predicting stroke disease has shown promising results in various studies.

Demerits:

- The need for large and diverse datasets to train the CNN effectively.
- Limited access to such datasets can hinder the model's performance and generalizability.

2.3.3 The Use of Deep Learning to Predict Stroke Patient Mortality, 2019

Authors: Songhee Cheon , Jungyoon Kim , Jihye Lim

Methodology:

The method used is DNN. This enhanced accuracy leads to more reliable predictions of mortality risks. Secondly, deep learning algorithms can detect subtle signs and risk factors for mortality at an early stage, enabling healthcare professionals to intervene promptly and provide targeted treatments. This early detection can significantly improve patient outcomes and reduce mortality rates. Additionally, deep learning models can be trained on large datasets, allowing for the development of personalized risk assessment tools that consider individual patient characteristics and medical histories. Lastly, once trained, deep learning models can be deployed at scale, potentially benefiting a large number of patients across different healthcare facilities.

However, there are several disadvantages associated with using deep learning for predicting stroke patient mortality. To begin, the accuracy and usefulness of deep learning models rely heavily on the quality and quantity of data utilized in their training. Poor-quality or biased datasets might result in inaccurate forecasts and possible dangers for patients. Second, deep learning models are frequently viewed as "black boxes," making it difficult to grasp their decision-making process. This lack of openness can undermine healthcare professionals' trust and impede the implementation of these models in clinical practice.Third, deep learning models necessitate significant computational capacity and machine learning skills, which can be expensive and present issues in healthcare settings with limited resources. Finally, ethical concerns about patient privacy, data security, and algorithmic biases must be addressed when employing deep learning in healthcare. Adherence to ethical rules and legislation is critical for guaranteeing the responsible use of deep learning to forecast stroke patient mortality.

Merits:

- Deep learning models can achieve higher accuracy compared to traditional methods by effectively analysing complex patterns and relationships within medical data.
- This enhanced accuracy leads to more reliable predictions of mortality risks.

Demerits:

- The accuracy is low of 84%
- Requires High time

2.3.4Deep learning-based personalised outcome prediction after acute ischaemic stroke, 2023

Authors: Doo-Young Kim, Kang-Ho Choi, Ja-Hae Kim, Jina Hong, Seong-Min Choi, Man-Seok Park, Ki-Hyun Cho

Methodology:

The study included 8,590 individuals with acute ischemic stroke (AIS) who were admitted within five days of symptom onset. The primary outcome was the number of major adverse cardiovascular events (MACEs), which comprised stroke, acute myocardial infarction, and death over a 12-month period. The study assessed the performance of deep learning models (DeepSurv and Deep-Survival-Machines) against traditional survival models (Cox proportional hazards and random survival forest) using the time-dependent concordance index (c-index).

Merits:

- Improved Predictive Accuracy and Deep learning models, particularly DeepSurv and Deep-Survival-Machines (DeepSM), demonstrate superior predictive performance compared to traditional survival models like Cox proportional hazards (CoxPH) and random survival forest (RSF).
- The incorporation of clinical data and brain imaging features enhances the accuracy of long-term risk prediction for MACE post-AIS.

Demerits:

- Data Complexity and Quality where Deep learning models require large, high-quality datasets for training, particularly when incorporating complex data sources like brain imaging.
- Ensuring data completeness, accuracy, and consistency can be challenging and may limit the generalizability of model findings.

2.3.5 : A deep learning system for retinal vessel calibre improves cardiovascular risk prediction in Asians with chronic kidney disease, 2023

Authors: Cynthia Ciwei Lim, Crystal Chong, Gavin Tan, Chieh Suai Tan, Carol Y Cheung, Tien Y Wong, Ching Yu Cheng, and Charumathi Sabanayagam

Methodology:

The Singapore Epidemiology of Eye Diseases Study (2004-2011) included 860 participants aged 40-80 years from Chinese, Malay, and Indian ethnic groups with chronic kidney disease (CKD) defined as an estimated glomerular filtration rate (eGFR) of less than 60 ml/min/1.73 m^2 at baseline. Retinal vascular calibre measurements were estimated using a deep learning technique on baseline retinal pictures. Participants were tracked until December 31, 2019, for the development of new cardiovascular disease (CVD) events such as non-fatal acute myocardial infarction (MI), stroke, and death from MI, stroke, or other CVD causes in those without prior CVD.Cox proportional hazards regression models were used to investigate the relationship between risk variables (conventional cardiovascular risk factors, eGFR, and retinal characteristics such as arteriolar constriction and venular dilation) and the development of CVD. The models were evaluated for discrimination, fit, and net reclassification improvement (NRI) to determine how adding kidney function and retinal vessel calibre measures to traditional cardiovascular risk factors affects CVD risk prediction in CKD patients, particularly in Asian populations. The findings highlight the feasibility of employing automated retinal vascular calibre measures to improve CVD risk prediction and classification in CKD patients.

Merits:

- Utilized a large and diverse cohort of 860 participants representing Chinese, Malay, and Indian ethnicities, enhancing generalizability.
- Employed a deep learning system for automated retinal vessel calibre measurements, facilitating efficient and objective assessment of retinal features.

Demerits:

Retrospective study design limits the ability to establish causal relationships between retinal vessel calibre and incident CVD.

Potential for selection bias due to participants attending a baseline visit of an eye diseases study, potentially impacting generalizability.

Reliance on estimated glomerular filtration rate (eGFR) <60 ml/min/1.73 m^2 as the sole criterion for CKD diagnosis may not capture all aspects of kidney function variability.

2.3.6 Deep Learning Improves Prediction of Cardiovascular disease (cvd), 2022

Authors: by Seung-Jae Lee, ORCID,Sung-Ho Lee,Hyo-In Choi 1ORCID,Jong-Young Lee, Yong-Whi Jeong,Dae-Ryong Kang, andKi-Chul Sung,ORCID

Methodology:

The study compares two models for predicting cardiovascular disease (CVD) outcomes in patients with hypertension: a conventional logistic regression model and a deep learning model1. The deep learning model demonstrated superior performance over the logistic regression model in predicting both CVD hospitalization and mortality. Utilized korean national health insurance service database

Merits:

- The deep learning model include its ability to handle large-scale, time-dependent datasets and its higher accuracy in risk stratification, which can significantly aid in early detection and resource allocation for high-risk patients.

Demerits:

- The complexity of the model, which can make it difficult to understand and trust, and the potential challenges in explaining individual algorithmic decisions, which may limit the ability to provide specific recommendations for controlling risk factors.
- Overall, the deep learning approach offers a promising tool for improving CVD outcome predictions, but with considerations for its interpretability and application in clinical practice.

2.3.7 Development of Deep Learning Models for Predicting In-Hospital Mortality Using an Administrative Claims Database: Retrospective Cohort Study

Authors: Hiroki Matsui, MPH, Hayato Yamana, MD, PhD, Kiyohide Fushimi, MD, PhD, and Hideo Yasunaga, MD, PhD

Methodology:

The work, which employed administrative claims data from the Japanese Diagnosis Procedure Combination dataset to develop and validate predictive models for in-hospital mortality in acute care patients, is summarised in the abstract. The primary model is trained using deep learning techniques, such as a 9-layer deep neural network, using patient demographics, diagnoses, and procedures noted at admission. Furthermore, condition-specific variables may be included in models designed for diseases such as acute myocardial infarction, heart failure, stroke, and pneumonia. Metrics like AUC and calibration plots are used to evaluate the discrimination and calibration of the model, and a different cohort is used for validation. The study's overall goal is to demonstrate how well deep learning predicts in-hospital mortality and offers guidance for medical professionals making decisions about patients' care.

Merits:

The Real-World Relevance As the data are derived from routine clinical practice, the predictive models developed using this dataset may have high external validity and real-world relevance.

Potential for Improved Accuracy and Efficiency of Deep learning algorithms can analyze vast amounts of medical data, potentially leading to more accurate diagnoses and treatment plans compared to traditional methods.

Demerits:

- **Data Limitations** is the review acknowledges; a major challenge is the availability of high-quality, comprehensive stroke data.
- Deep learning models can be complex, and their decision-making processes not easy.

CHAPTER 3

SYSTEM DIAGRAMS

3.1 SYSTEM ARCHITECTURE:

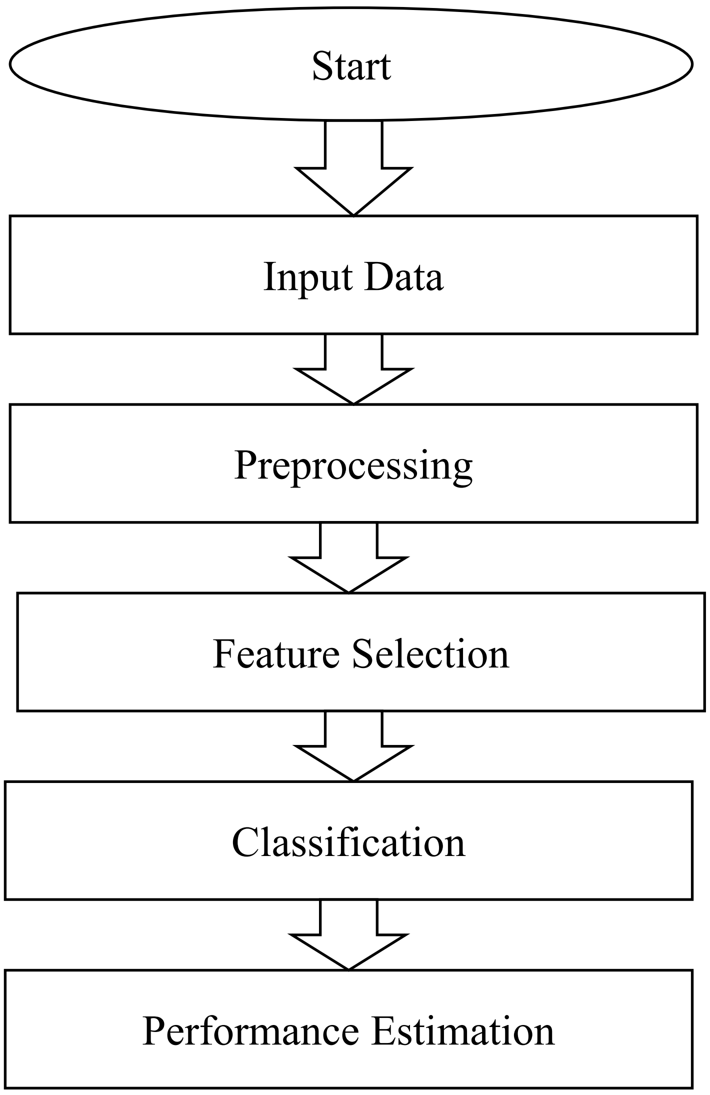


FIGURE 1: SYSTEM ARCHITECTURE

3.2 FLOW DIAGRAM

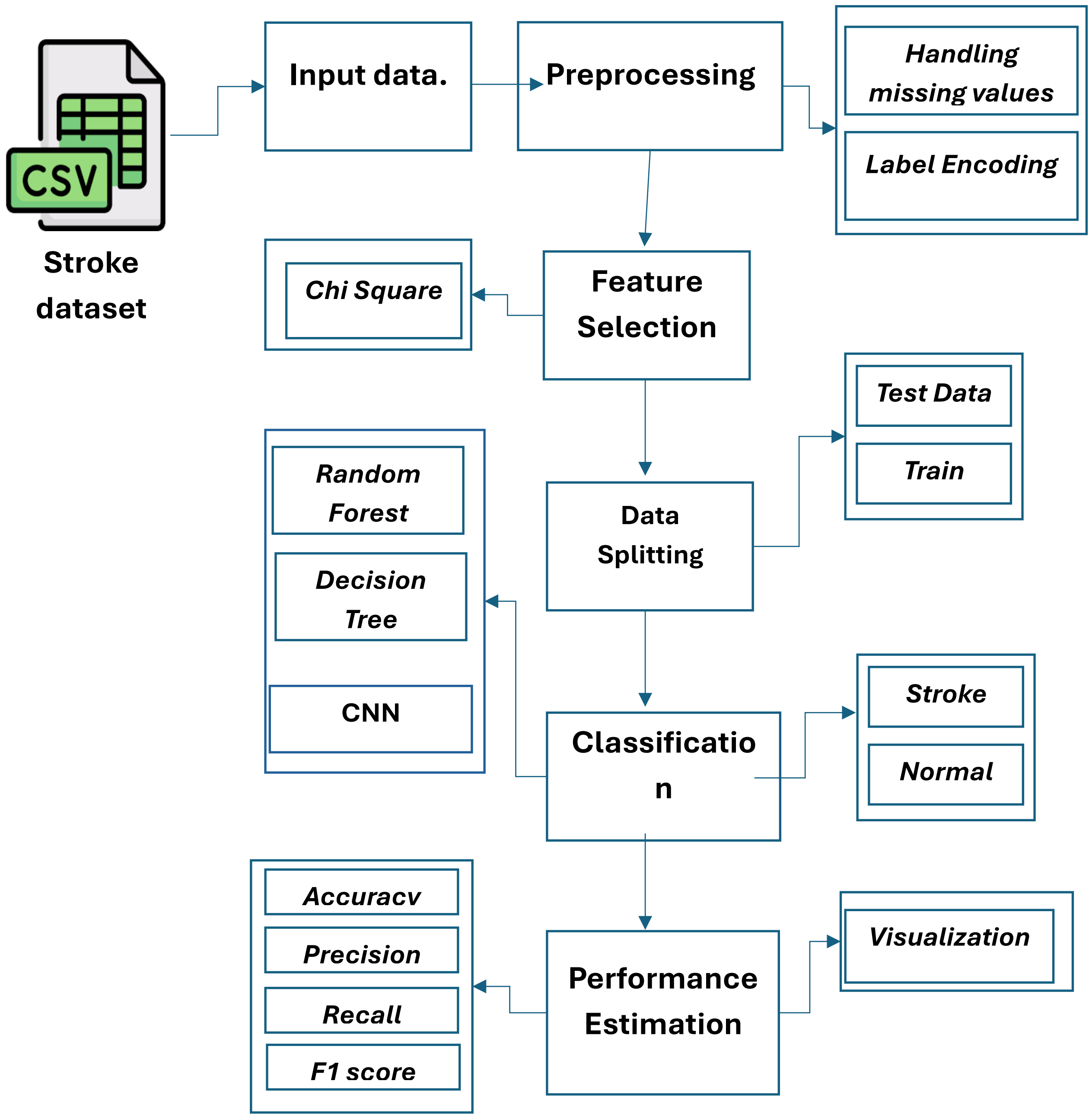


FIGURE 2: FLOW DIAGRAM

3.3 UML DIAGRAMS:

3.3.1 USE CASE DIAGRAM

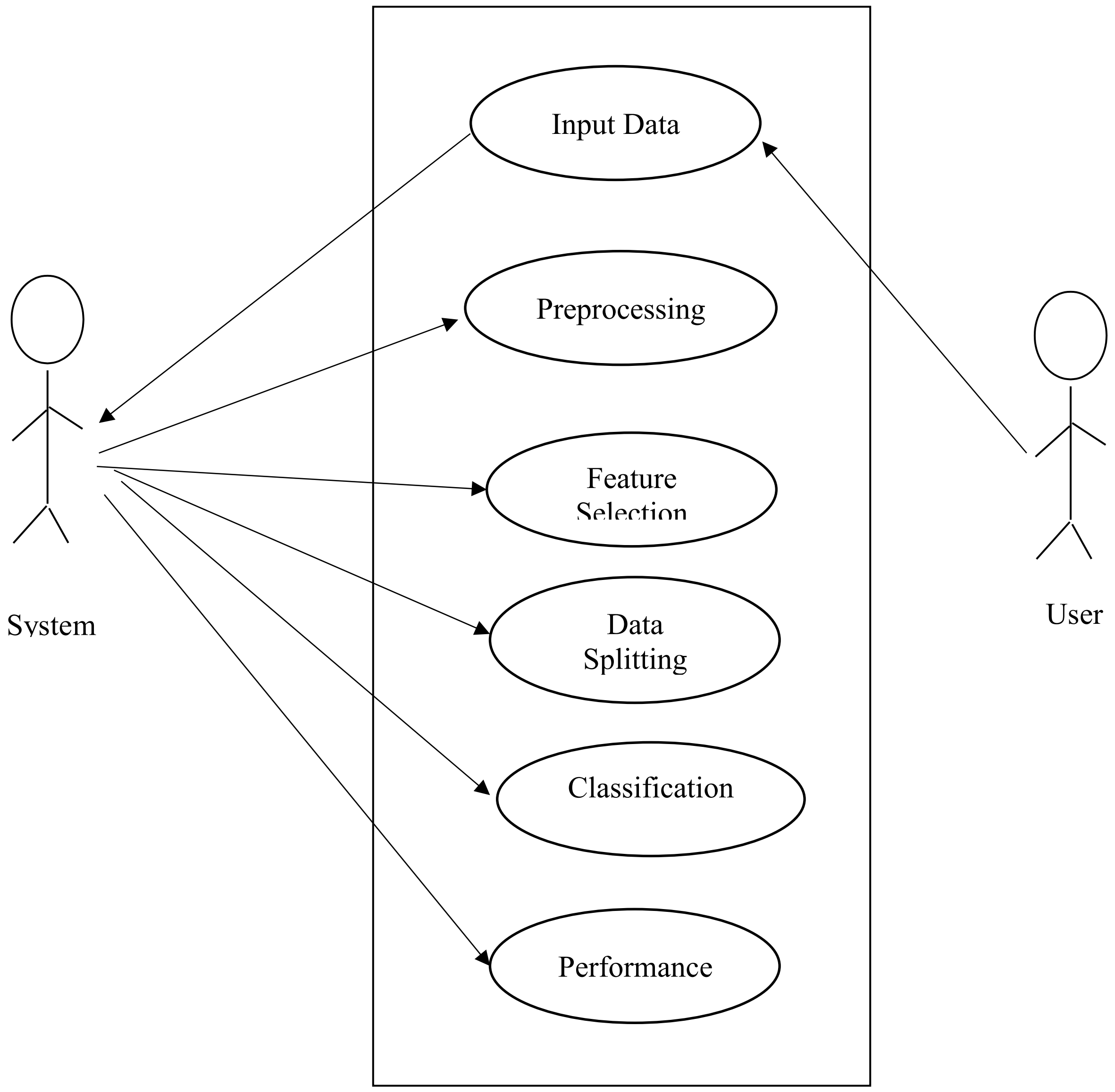


FIGURE 3: USE CASE DIAGRAM

3.3.2 ACTIVITY DIAGRAM:

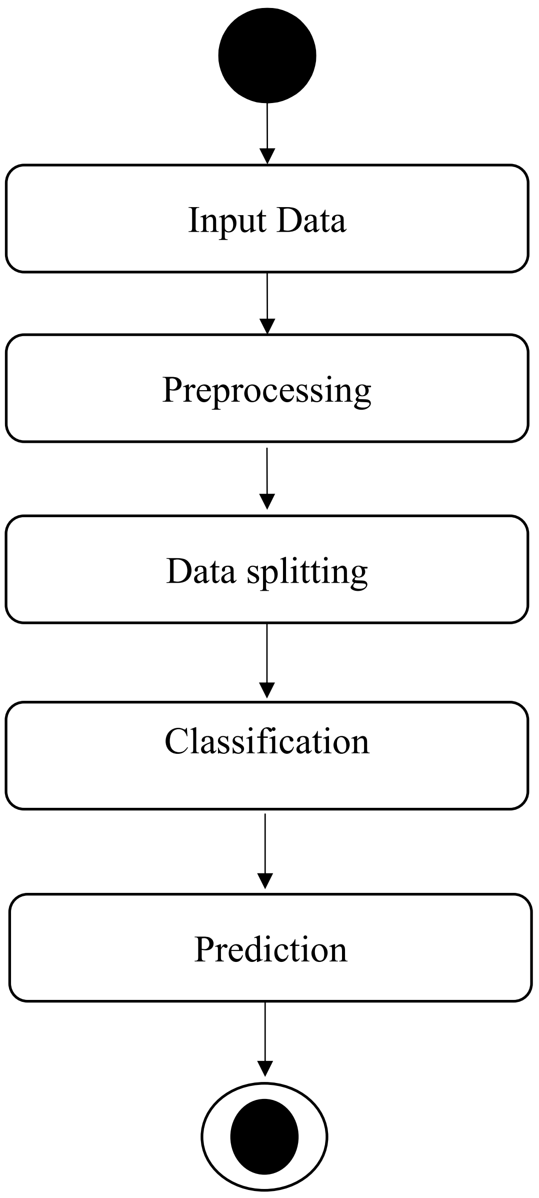


FIGURE 4: ACTIVITY DIAGRAM

3.3.3 SEQUENCE DIAGRAM:

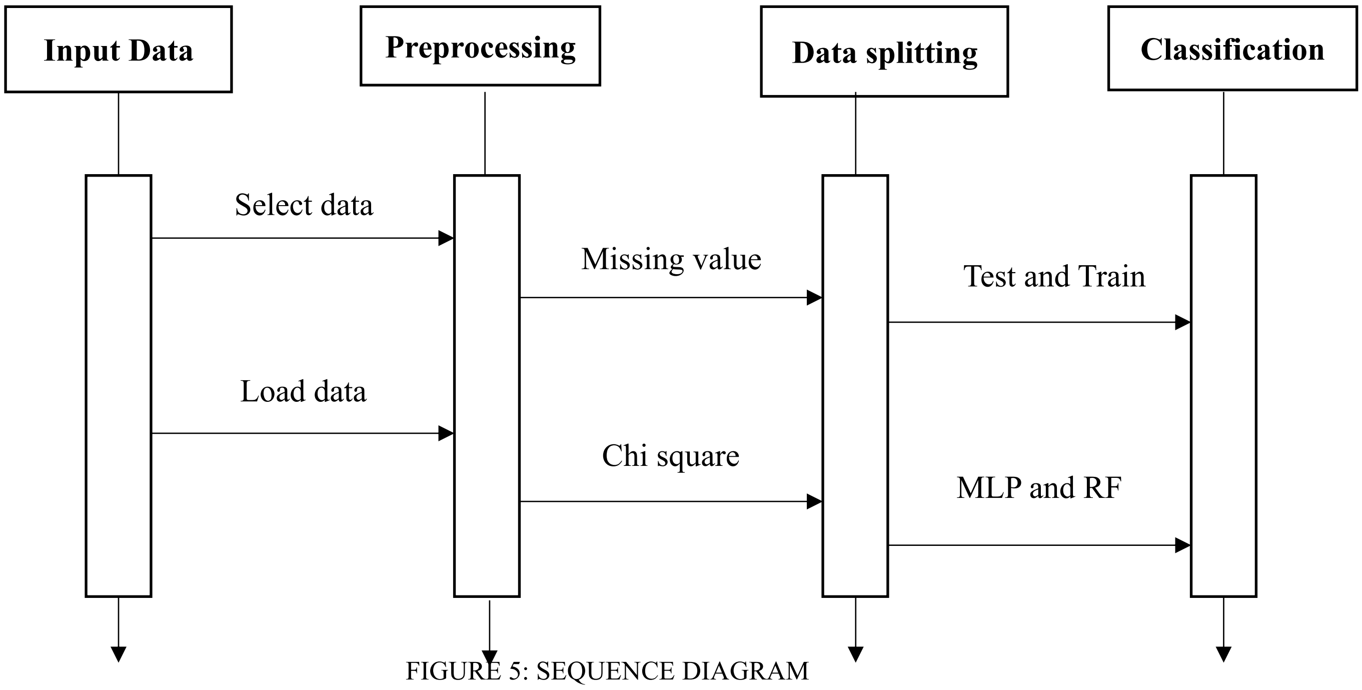


FIGURE 5: SEQUENCE DIAGRAM

3.3.4 ER DIAGRAM:

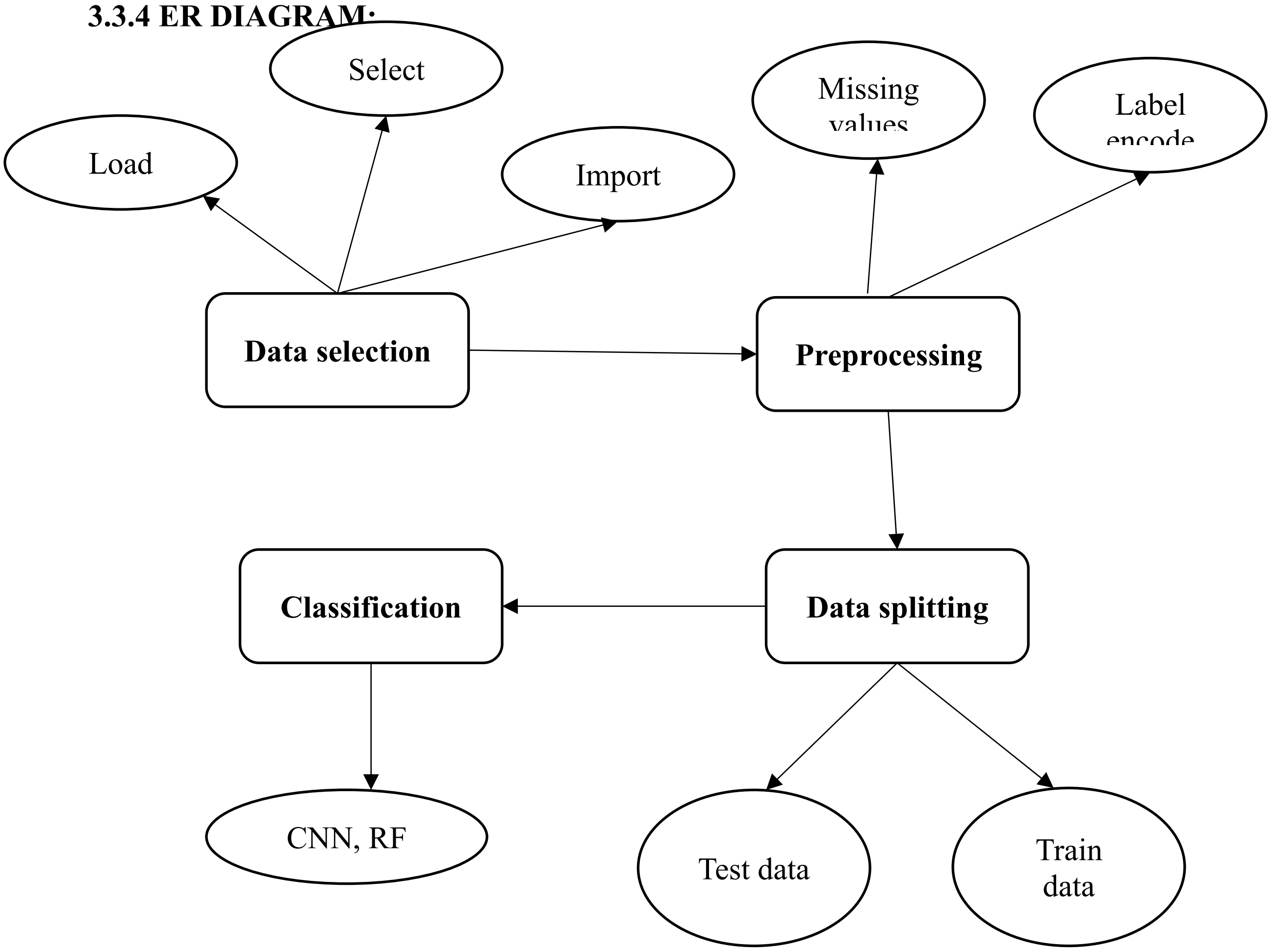


FIGURE 6: ER DIAGRAM

3.3.6 CLASS DIAGRAM:

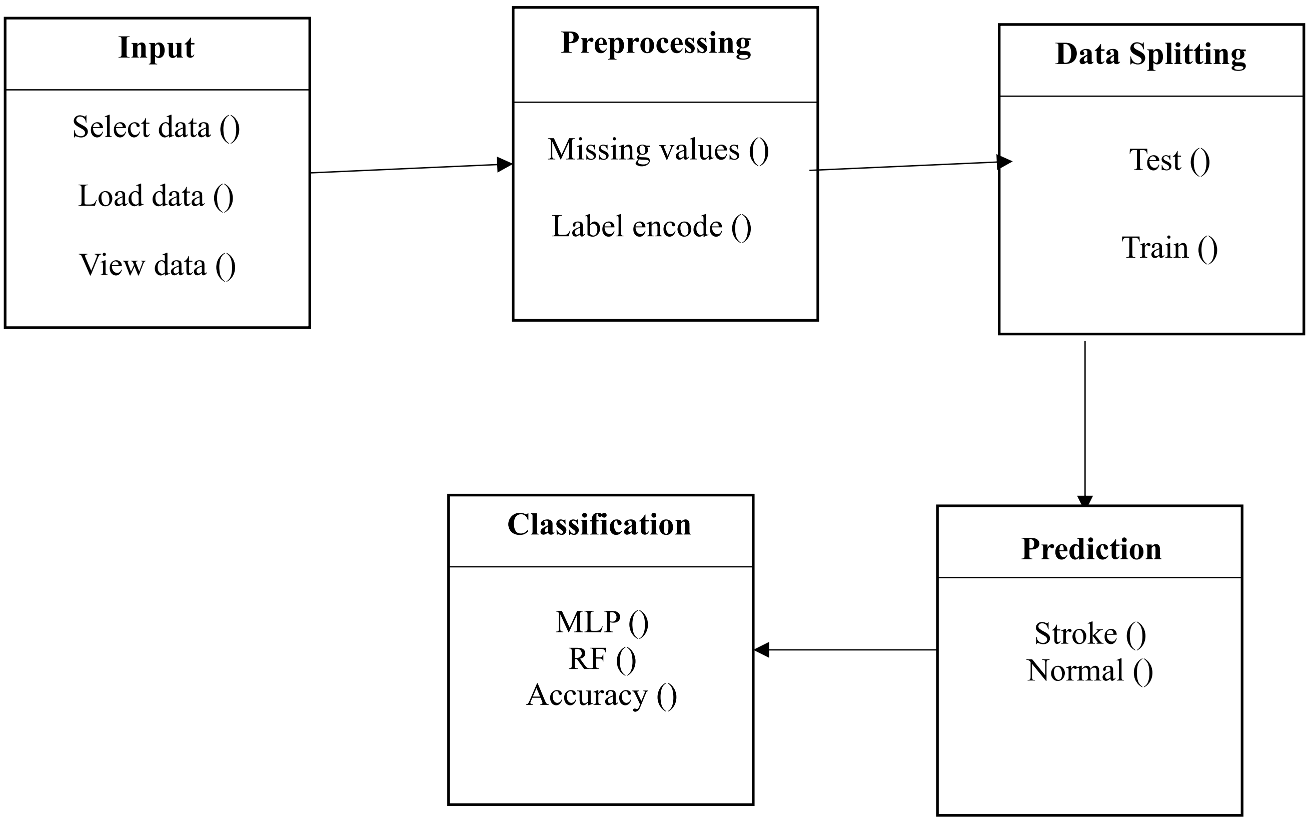


FIGURE 7: CLASS DIAGRAM

CHAPTER 4

4.1 ABOUT THE MODEL USED IN THE PROJECT:

1. **Logistic Regression (reg)**:
   - **Type**: Classification o **Description**: A linear model used for binary classification problems is called logistic regression.
   - It calculates the likelihood that an input falls into a specific category.
   - **Parameters**: max_iter and class_weight are tuned to control the maximum number of iterations for optimization and to balance the class weights respectively.

2. **Support Vector Classifier (svc)**:
   - **Type**: Classification

- **Description**: Support Vector Machine (SVM) constructs hyperplanes in a highdimensional space to separate different classes. SVC is the classification variant of SVM.
- **Parameters**: Default parameters are used in this code.

3. **Decision Tree Classifier (dtc)**:
   - **Type**: Classification
   - **Description**: Decision Trees use fundamental rules of decision-making deduced from the data attributes to build a model that forecasts the target variable.
   - **Parameters**: Default parameters are used.

4. **Random Forest Classifier (rfc)**:
   - **Type**: Classification
   - **Description**: During training, many decision trees are generated using the Random Forest ensemble learning technique, which yields a class that is the mean of the classes of individual trees.
   - **Parameters:** The variables max_depth, min_samples_split, max_estimators, and max_features control the maximum depth of the trees, the amount of features to consider when determining the best split, and the least number of samples needed to split an internal node, respectively.

5. **Gradient Boosting Classifier (gbc)**:
   - **Type**: Classification
   - **Description**: Gradient Boosting allows the optimization of any differentiable loss function by building an additive model in a forward, step-by-step manner.
   - **Parameters**: Default parameters are used.
6. **AdaBoost Classifier (abc)**:
   - **Type**: Classification
   - **Description**: AdaBoost (Adaptive Boosting) focuses on classification problems and functions by assigning a weight to each example in the dataset according on how simple or complex it is to classify, enabling the algorithm to give each instance more or less attention during training.
   - **Parameters**: Default parameters are used.
7. **K-Nearest Neighbours Classifier (KNC)**:

- **Type**: Classification
- **Description**: K-Nearest Neighbours (KNN) is a non-parametric lazy learning method that categorizes a data point according to the classification of its neighbours.
- It uses the Euclidean distance to find the K nearest neighbours.
- **Parameters**: K near neighbours is tuned to control the number of neighbours considered.

4.2 INFORMATION ABOUT THE ARCHITECTURE:

Certainly! Here's a rewritten architecture tailored to the provided code for stroke prediction:

1. Data Loading:

The initial step involves loading the stroke prediction dataset containing various health metrics and attributes of individuals, including stroke occurrence. This dataset is sourced from a CSV file and is crucial for subsequent analysis and modeling.

2. Preprocessing:

After loading the dataset, Preprocessing measures are implemented to guarantee the data is suitable for model training. This includes handling missing values by dropping rows with NaN values, removing irrelevant columns like 'id', encoding categorical variables into numerical format using LabelEncoder, and removing outliers using the Interquartile Range (IQR) method.

3. Data Visualization:

Exploratory Data Analysis (EDA) is executed through data visualization to comprehend the distribution of features. This involves plotting histograms, pie charts, and scatterplots to visualize the distribution of individual features, relationships between variables, and the target variable ‘stroke' distribution.

4. Feature Scaling and Splitting:

The dataset is divided into training and testing sets prior to model training. Additionally, feature scaling is applied to standardize the feature values using the StandardScaler to ensure consistent and optimal model performance.

5. Model Building:

The model building component, which is the central component of the design, is where different classification algorithms are taught to anticipate strokes. Logistic regression, Support

Vector Machine (SVM), Random Forest, Decision Tree, Gradient Boosting, AdaBoost, and K-Nearest Neighbors classifiers are among the models.

6. Model Training and Evaluation:

The models are trained on the training dataset and subsequently tested on the testing dataset. A model's performance is assessed using measures including F1 score, accuracy, precision, and recall. Hyperparameter tweaking is also utilized to optimize the model's performance in different configurations.

7. Model Comparison:

Scores of the trained models are compiled into a DataFrame for comparison, helping to identify the best-performing model for stroke prediction.

8. ROC and Precision-Recall Curves:

ROC and Precision-Recall curves are plotted for each classifier to visualize their performance and determine the Area Under the Curve (AUC) and average precision (AP), respectively.

9. Feature Importance:

For the Random Forest Classifier and Gradient Boosting Classifier models, feature importance is plotted to identify the top features contributing to stroke prediction.

10. Inference:

After selecting the best-performing model, it can be deployed for real-time inference on new data. The deployed model will predict the likelihood of stroke occurrence based on input health metrics, assisting in early detection and preventive healthcare measures.

This architecture outlines the comprehensive process of stroke prediction using machine learning, encompassing data loading, preprocessing, exploratory analysis, model building, training, evaluation, and inference.

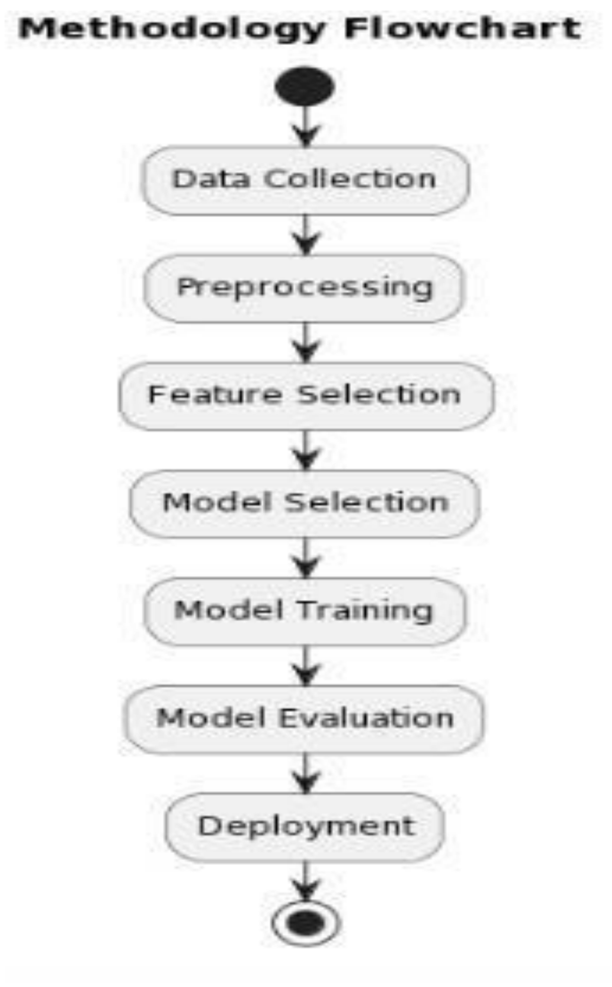
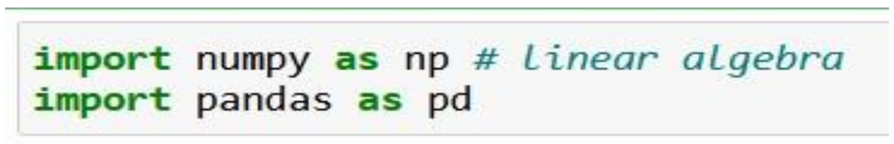


Figure 8: Methodology flowchart

IMPLEMENTATION OF THE PROJECT:

IMPORTING MODULES REQUIRED:

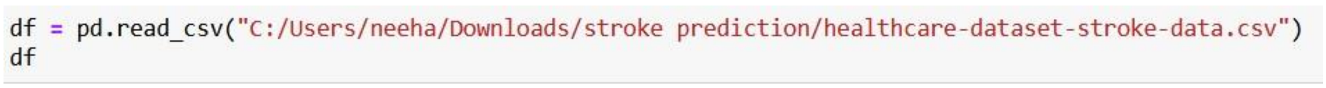


```
import numpy as np # linear algebra
import pandas as pd
```

NumPy: A library called NumPy adds support for massive multidimensional arrays and matrices to the Python programming language. It also provides a vast variety of high-level mathematical functions that may be used to manipulate these arrays. It's a fundamental package for scientific computing with Python.

Pandas: Built on top of NumPy, Pandas offers functions and data structures that make dealing with structured data quick, simple, and expressive. It is particular for well-suited in handling tabular data, such as data from spreadsheets or databases, and provides tools for reading, writing, and manipulating this data.

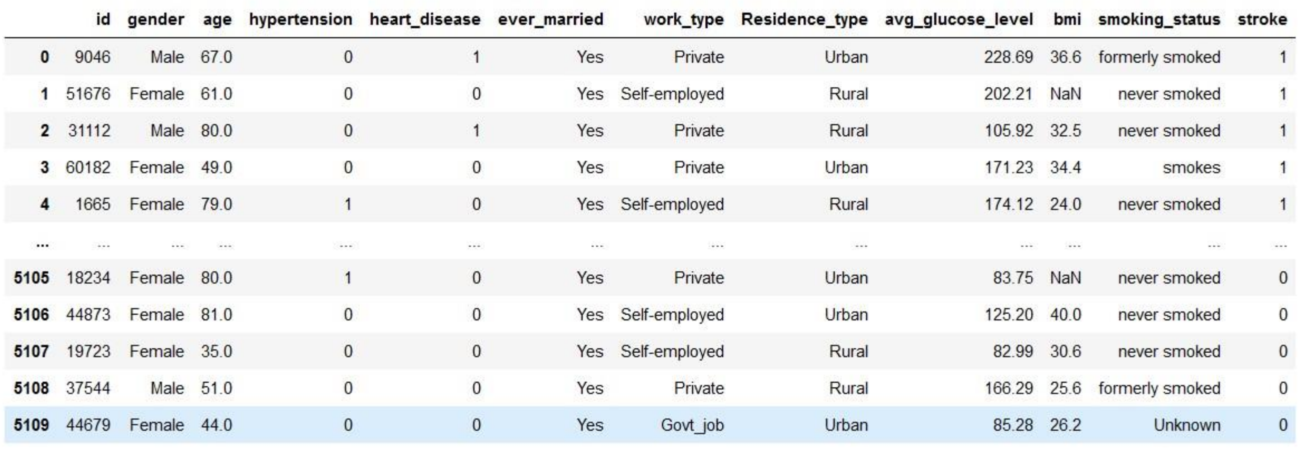
LOADING THE DATASET:



```
df = pd.read_csv("C:/Users/neeha/Downloads/stroke prediction/healthcare-dataset-stroke-data.csv")
df
```

This code reads a CSV (Comma Separated Values) file using Pandas and store the data information in a Data Frame object named df.pd.read_csv("C:/Users/neeha/Downloads/stroke prediction/healthcare-dataset-stroke-data.csv"): This function call reads the CSV file located at the specified path ("C:/Users/neeha/Downloads/stroke prediction/healthcare-dataset-stroke-data.csv") and converts it into a Pandas DataFrame. fThe read_csv() function is a part of Pandas and is used to read CSV files into DataFrames.

df: This is the variable name assigned to the DataFrame that stores the data and information from the CSV file. You can use that variable to access and manipulate the data in the DataFrame throughout your code.



	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
...	...	...	...	...	...	...	...	...	...	...	...	...
5105	18234	Female	80.0	1	0	Yes	Private	Urban	83.75	NaN	never smoked	0
5106	44873	Female	81.0	0	0	Yes	Self-employed	Urban	125.20	40.0	never smoked	0
5107	19723	Female	35.0	0	0	Yes	Self-employed	Rural	82.99	30.6	never smoked	0
5108	37544	Male	51.0	0	0	Yes	Private	Rural	166.29	25.6	formerly smoked	0
5109	44679	Female	44.0	0	0	Yes	Govt_job	Urban	85.28	26.2	Unknown	0

INFO AND FEATURES OF DATASET: Pandas' info() function offers a concise a description of the Data Frame that includes the names of the columns, the types of data they include, the amount of non-null values, and the amount of RAM used. Range Index: This shows the Data Frame’s index's range.

.

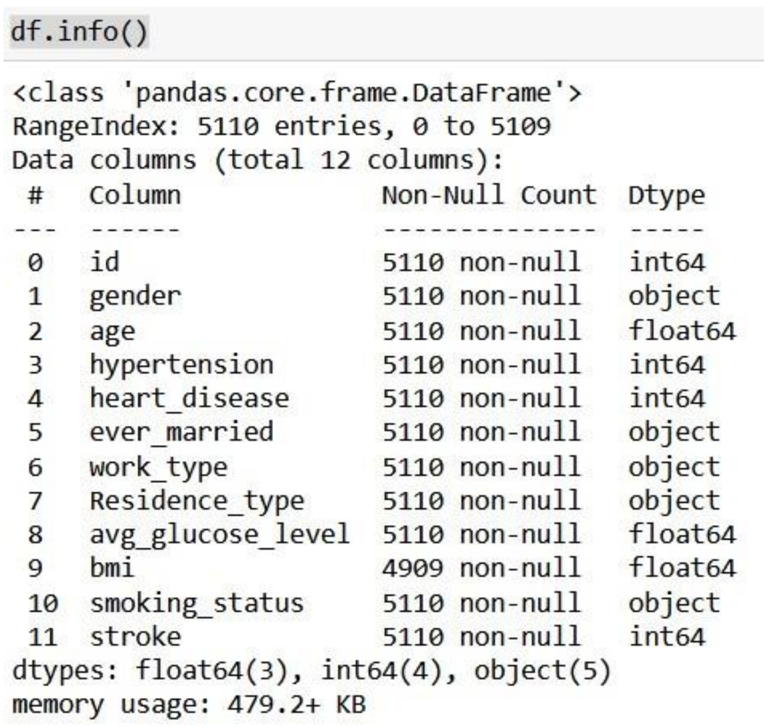
Columns: Lists the names of all columns in the DataFrame.

Non-Null Count: Shows many number of non-null (non-missing) values in each and every column. This can be useful for identifying missing data.

Dtype: shows the type of data for each column. (e.g., int64, float64, object).

Memory Usage: It Shows the memory usage of a DataFrame.

The info() method is particularly helpful when you first load a dataset to understand its structure, the types of data it contains, and whether there are any missing values. This summary can guide you in further data preprocessing and analysis tasks.



```
df.info()
```

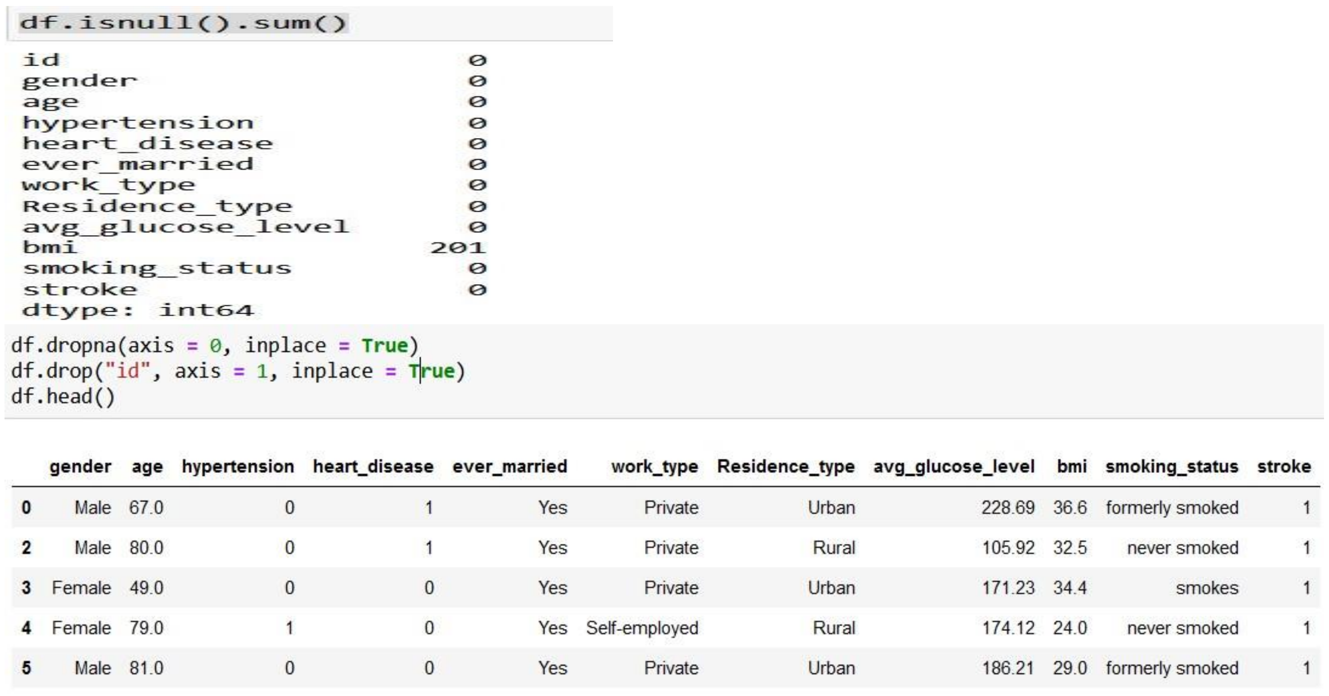
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5110 non-null   int64
 1   gender             5110 non-null   object
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64
 4   heart_disease      5110 non-null   int64
 5   ever_married       5110 non-null   object
 6   work_type          5110 non-null   object
 7   Residence_type     5110 non-null   object
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object
 11  stroke             5110 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

PREPROCESSING OF DATA:

This code computes Utilizing the isnull() function in conjunction with the sum() method, determine the amount of missing values (null values) in each column of the DataFrame df. df.isnull(): This function yields a DataFrame with the same structure as df, where each element is False otherwise and True when comparable elements in df are NaN (null).

.sum(): This method then sums up the boolean values along each column, resulting in a Series where each value represents the number of null values total in the corresponding column.

By executing df.isnull().sum(), you get a Series object with values denoting the amount of missing data in each column and index representing the names of the columns. This information could be useful for identifying and handling missing data in your dataset.



```
df.isnull().sum()
```

```
id                    0
gender                0
age                   0
hypertension          0
heart_disease         0
ever_married          0
work_type             0
Residence_type        0
avg_glucose_level     0
bmi                 201
smoking_status        0
stroke                0
dtype: int64
```

```
df.dropna(axis = 0, inplace = True)
df.drop("id", axis = 1, inplace = True)
df.head()
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
2	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
5	Male	81.0	0	0	Yes	Private	Urban	186.21	29.0	formerly smoked	1

1. `df.dropna(axis=0, inplace=True)`: This line drops rows with any missing values (NaNs) along the rows (axis 0) from the DataFrame `df`. The parameter `inplace=True` ensures that the changes are applied directly to `df` without returning a new DataFrame.

2. `df.drop("id", axis=1, inplace=True)`: This line drops the column named "id" from the DataFrame `df` along the columns (axis 1). Again, `inplace=True` ensures that the change is applied directly to
`df`.

After executing these operations, the DataFrame `df` will have rows with any missing values removed, and the column "id" will be dropped. Finally, `df.head()` shows the first some rows of the modified
DataFrame `df`.

This code generates a series of plots to visualize the distribution of data in each column of the DataFrame `df`.

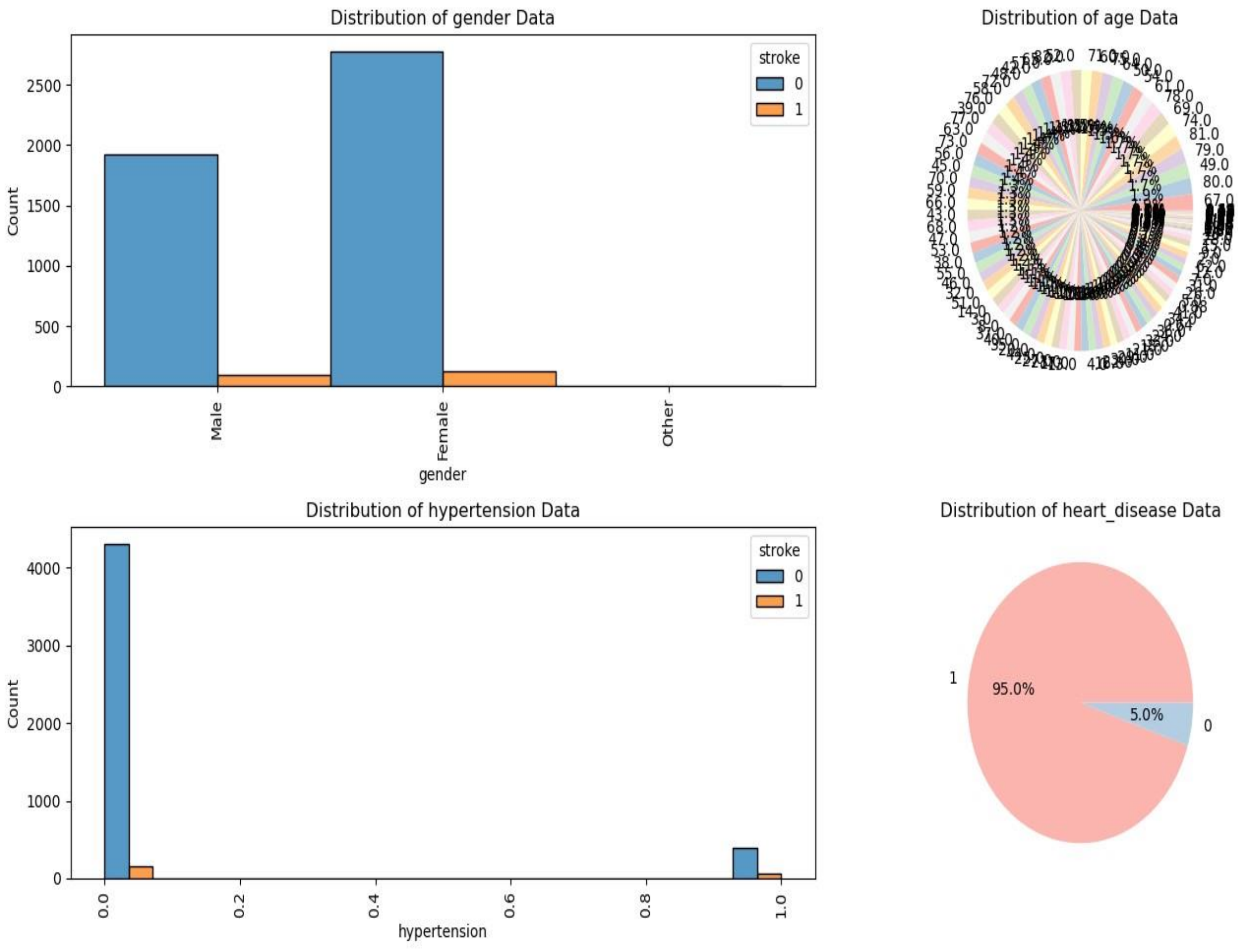


Figure 9: Distribution of data

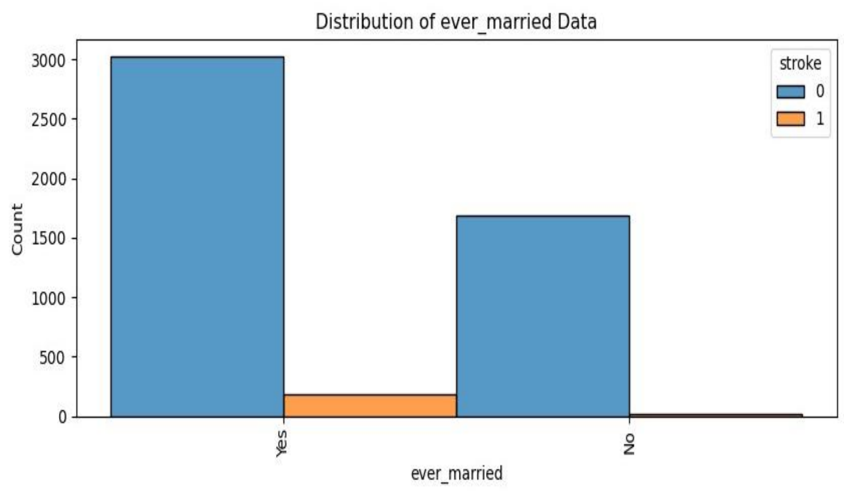
1. It imports the necessary libraries: `warnings`, `matplotlib.pyplot` (as `plt`), and `seaborn` (as `sns`).
2. It sets up a warning filter to ignore warnings.

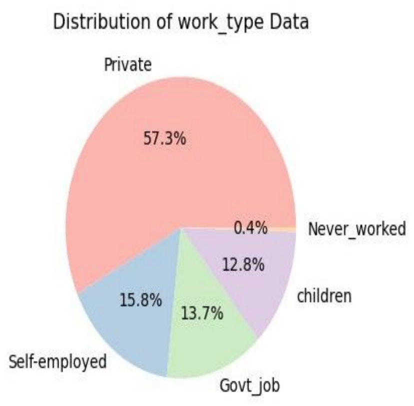
3. It sets up the plotting environment and initializes a figure with a size of 15x20 inches.

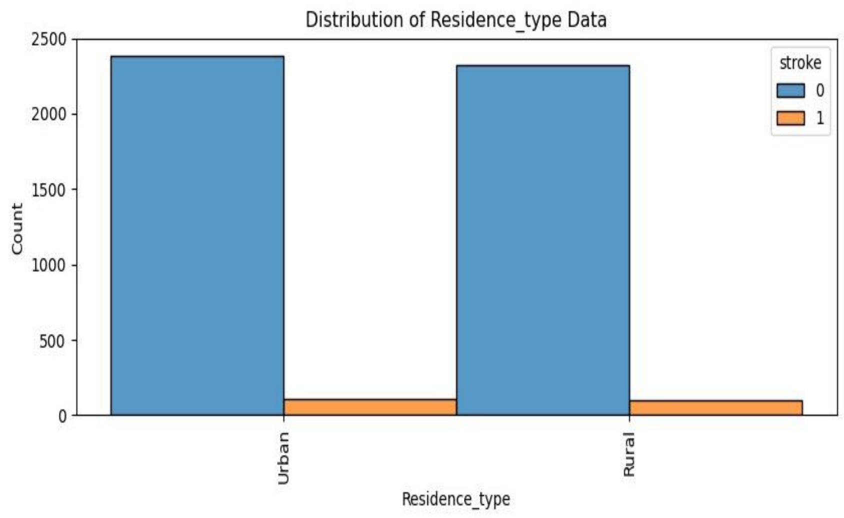
4. It iterates over each column in the DataFrame (`df.columns[:-1]` excludes the last column, assuming it's the target variable).

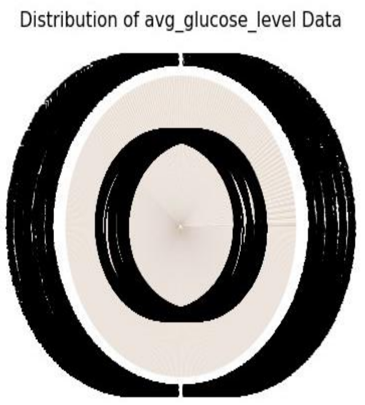
5. For each column, it checks if its index `i` is even or odd:

- If `i` is even, it creates a pie chart showing the distribution of unique values in that column. - If `i` is odd, it creates a histogram showing the distribution of values in that column, with bars colored based on the "stroke" column (assuming "stroke" is the target variable).



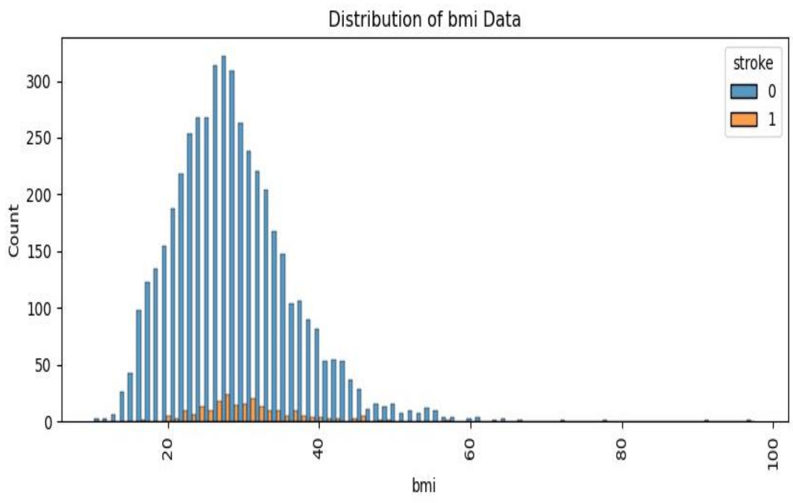


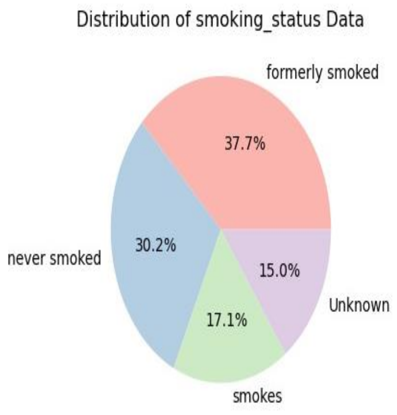




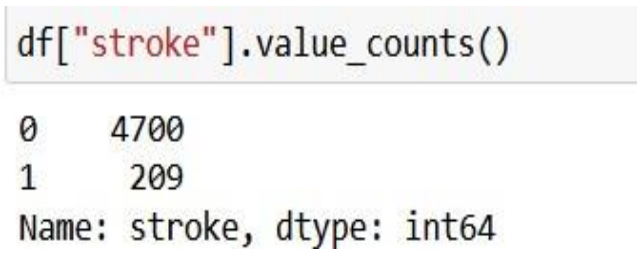
6. It sets titles for each subplot based on the column name.

7. It adjusts the layout, rotates x-axis labels if needed, and finally displays the plots.





This code provides a visual overview of the distribution of data in each column and how it relates to the "stroke" variable, helping to understand patterns and potential correlations in the data.

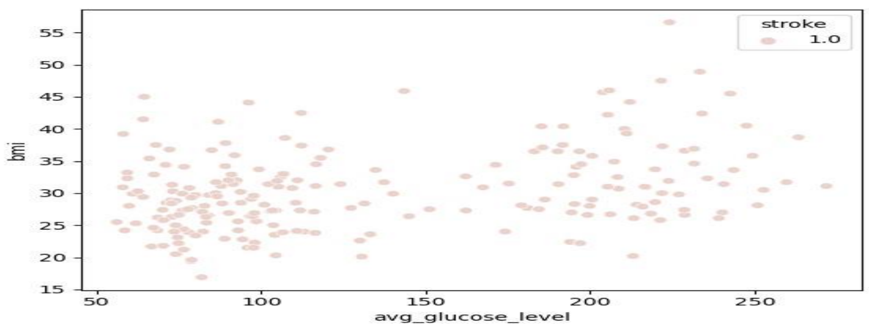


```
df["stroke"].value_counts()
```

```
0    4700
1     209
Name: stroke, dtype: int64
```

- The `value_counts()` function yields a Series with counts of distinct values.
- In this case, it counts occurrences of each and every unique values in the "stroke" column, which likely represents whether an individual had a stroke or not (e.g., 1 for stroke occurrence, 0 for no stroke).

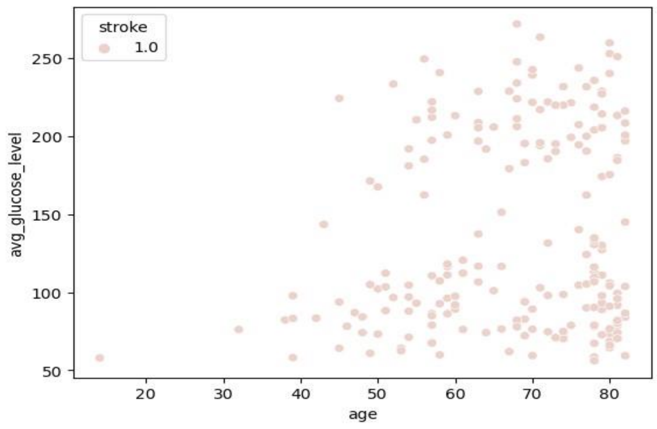
The result will show how many individuals in the dataset experienced a stroke (assuming the value 1 represents a stroke) and how many did not (assuming the value 0 represents no stroke). This information is very useful for understanding the balance of classes in the data set and is often crucial for tasks like classification or predictive modeling.



- Using Seaborn's `scatterplot()` function, this code generates a scatter plot to illustrate the correlation between the "avg_glucose_level" and "bmi" columns from the DataFrame. `df`. It also colors the points based on whether an individual had a stroke or not.
- `sns.scatterplot(x=df["avg_glucose_level"], y=df["bmi"], hue=df[df["stroke"] == 1]["stroke"])`: This line creates a scatter plot where the x-axis represents the "avg_glucose_level" column, the y-axis represents the "bmi" column, and the color of each point is determined by the "stroke" column. The hue parameter is set to `df[df["stroke"] == 1]["stroke"]`, which colors the points based on whether the value of "stroke" is equal to 1 (assuming 1 represents a stroke). This separates the points into two groups: those with strokes and those without strokes.

- `plt.show()`: This line displays the plot.

This scatter plot can be help to display any potential connection or clustering between average glucose levels, BMI, and the occurrence of strokes. It allows for a quick assessment of whether there are any discernible patterns or correlations between these variables.



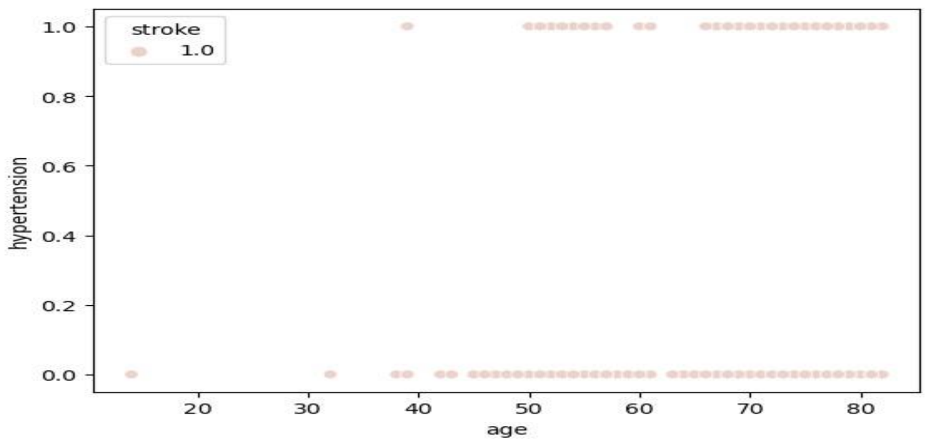
Using Seaborn's `scatterplot()` function, this code generates a scatter plot to show the relationship between the "age" and "avg_glucose_level" columns from the DataFrame `df`. It also colors the points based on whether an individual had a stroke or not.

Here's what each part of the code:

`sns.scatterplot(x=df["age"], y=df["avg_glucose_level"], hue=df[df["stroke"] == 1]["stroke"])`: This line creates a scatter plot where the x-axis represents the "age" column, the y-axis represents the "avg_glucose_level" column, and the color of each point is determined by the "stroke" column. The hue parameter is set to `df[df["stroke"] == 1]["stroke"]`, which colors the points based on whether the value of "stroke" is equal to 1 (assuming 1 represents a stroke). This separates the points into two groups: those with strokes and those without strokes.

- `plt.show()`: This line displays the plot.

This scatter plot can also help visualize any potential connection or clustering between age, average glucose levels, and the occurrence of strokes. It allows for a quick assessment of whether there are any discernible patterns or correlations between these variables.



This code creates a scatter plot using Seaborn's `scatterplot()` function to visualize the relationship between the "age" and "hypertension" columns from the DataFrame `df`. It also colors the points based on whether an individual had a stroke or not.

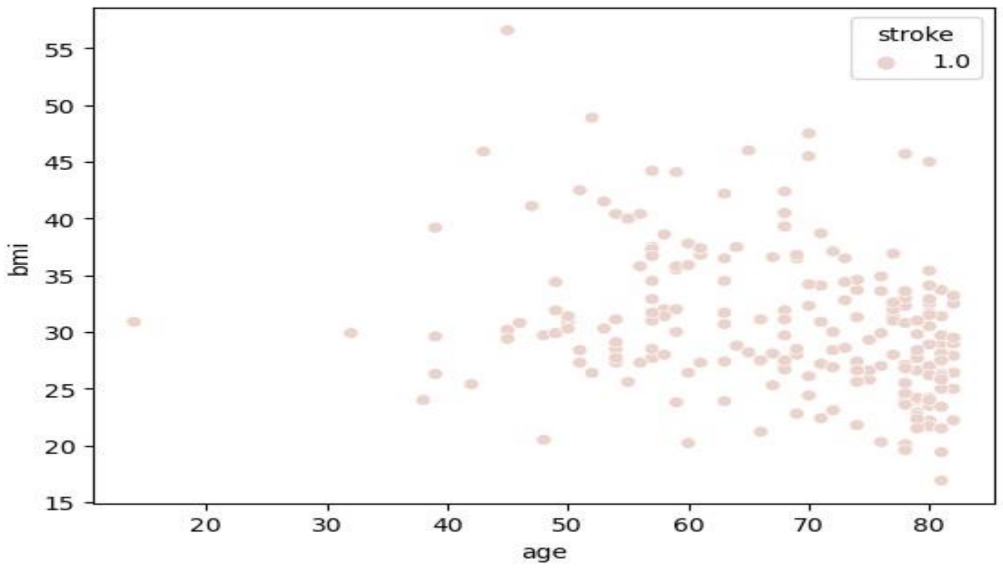
Here's what each part of the code does:

- `sns.scatterplot(x=df["age"], y=df["hypertension"], hue=df[df["stroke"] == 1]["stroke"])`: This line creates a scatter plot where the x-axis represents the "age" column, the y-axis represents the "hypertension" column, and the color of each point is determined by the "stroke" column. The hue parameter is set to `df[df["stroke"] == 1]["stroke"]`, which colors the points based on whether the value of "stroke" is equal to 1 (assuming 1 represents a stroke). This separates the points into two groups:

those with strokes and those without strokes.

- `plt.show()`: This line displays the plot.

This scatter plot can help visualize any potential relationship or clustering between age, hypertension, and the occurrence of strokes. It allows for a quick assessment of whether there are any discernible patterns or correlations between these variables.



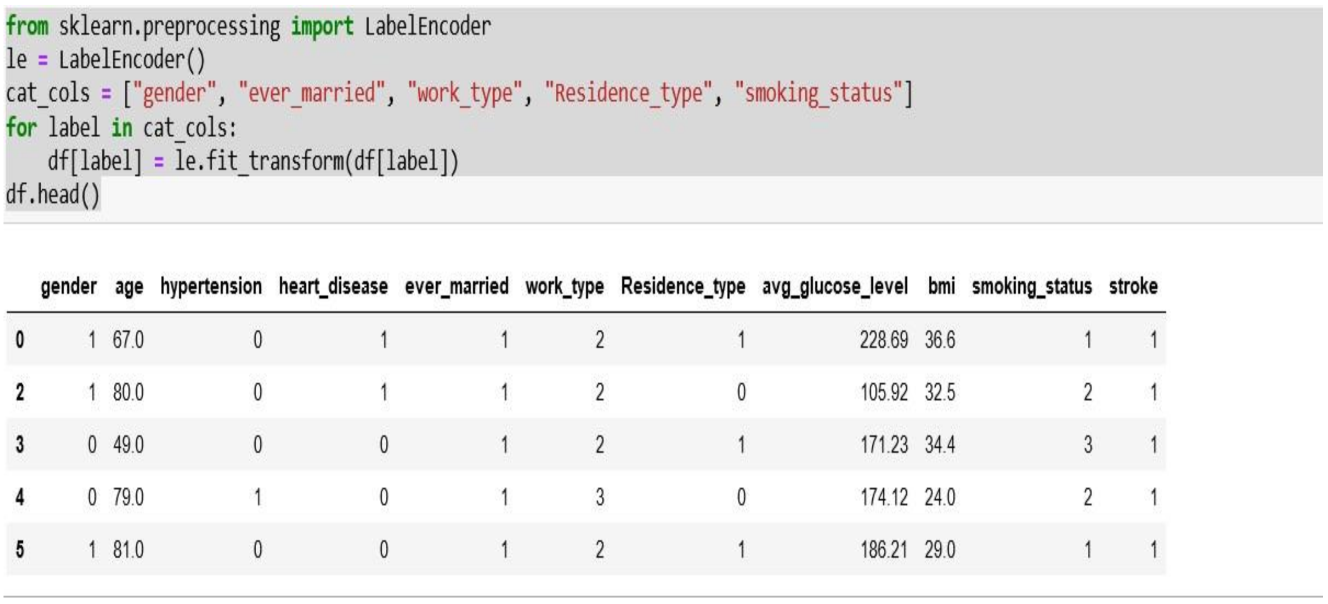
This code creates a scatter plot using Seaborn's `scatterplot()` function to visualize the relationship between the "age" and "bmi" columns from the DataFrame `df`. It also colors the points based on whether an individual had a stroke or not.

Here's what each part of the code does:

- `sns.scatterplot(x=df["age"], y=df["bmi"], hue=df[df["stroke"] == 1]["stroke"])`: This line creates a scatter plot where the x-axis represents the "age" column, the y-axis represents the "bmi" column, and the color of each point is determined by the "stroke" column. The hue parameter is set to `df[df["stroke"] == 1]["stroke"]`, which colors the points based on whether the value of "stroke" is equal to 1 (assuming 1 represents a stroke). This separates the points into two groups: those with strokes and those without strokes.

- `plt.show()`: This line displays the plot.

This scatter plot can help visualize any potential relationship or clustering between age, BMI (body mass index), and the occurrence of strokes. It allows for a quick assessment of whether there are any discernible patterns or correlations between these variables.



```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
cat_cols = ["gender", "ever_married", "work_type", "Residence_type", "smoking_status"]
for label in cat_cols:
    df[label] = le.fit_transform(df[label])
df.head()
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	1	67.0	0	1	1	2	1	228.69	36.6	1	1
2	1	80.0	0	1	1	2	0	105.92	32.5	2	1
3	0	49.0	0	0	1	2	1	171.23	34.4	3	1
4	0	79.0	1	0	1	3	0	174.12	24.0	2	1
5	1	81.0	0	0	1	2	1	186.21	29.0	1	1

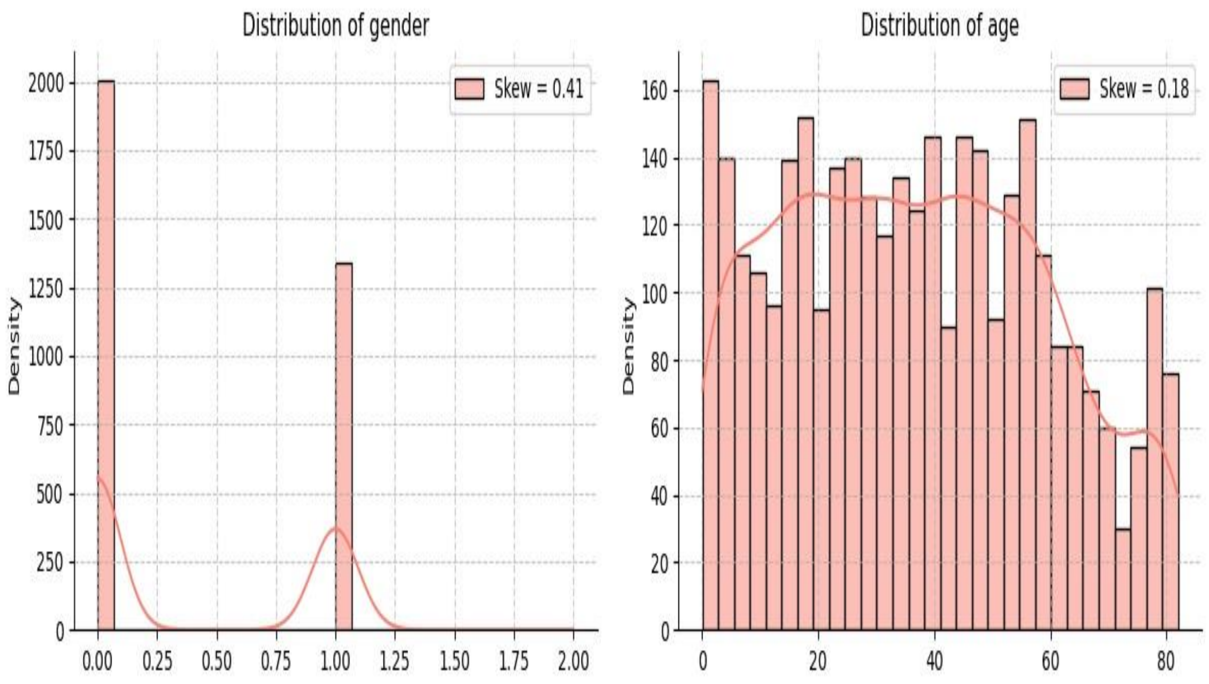
This code snippet encodes categorical variables in the DataFrame {df} into numerical labels by using the `LabelEncoder} class from the `sklearn.preprocessing` module.

Here's what each part of the code does:

` from sklearn.preprocessing import LabelEncoder}: This line imports the `sklearn.preprocessing` module's `LabelEncoder` class. To transform category labels into numerical labels, use {LabelEncoder}.

- `le = LabelEncoder()`: This line creates an instance of the `LabelEncoder` class, which will be used to encode categorical variables.
- `cat_cols = ["gender", "ever_married", "work_type", "Residence_type", "smoking_status"]`: This line defines a list `cat_cols` containing the names of categorical columns in the DataFrame `df` that need to be encoded.
- The `for` loop iterates over each categorical column name in `cat_cols`.
- `le.fit_transform(df[label])`: This line fits the `LabelEncoder` instance to the values of the current categorical column (`df[label]`) and transforms those values into numerical labels.
- `df[label] = ...`: This line assigns the transformed numerical labels back to the original column in the DataFrame.

Finally, `df.head()` visualize the first few rows of the DataFrame `df` after the categorical variables have been encoded into numerical labels. This transformation enables methods for machine learning to handle categorical data, which typically require numerical inputs.



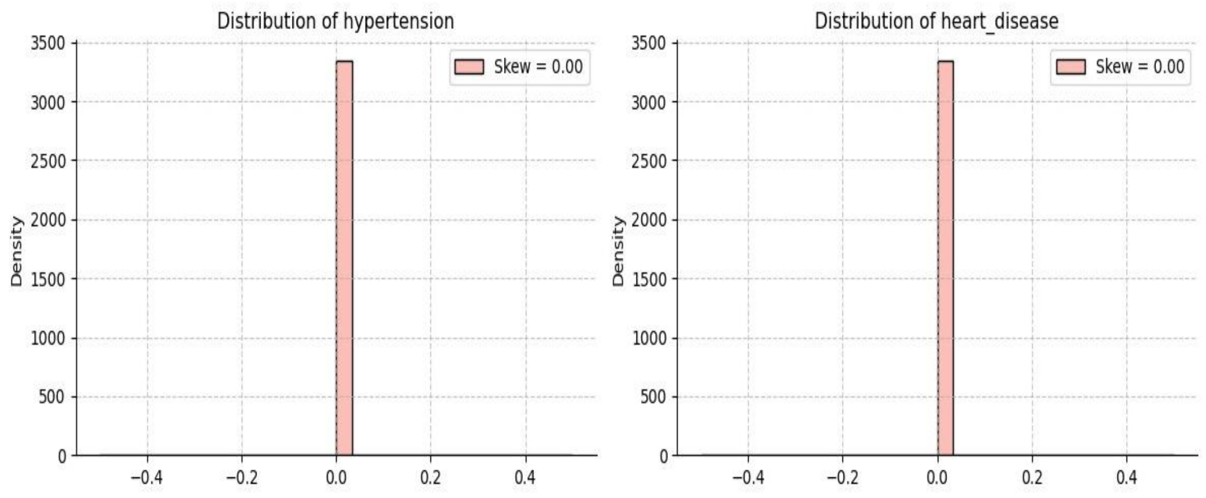
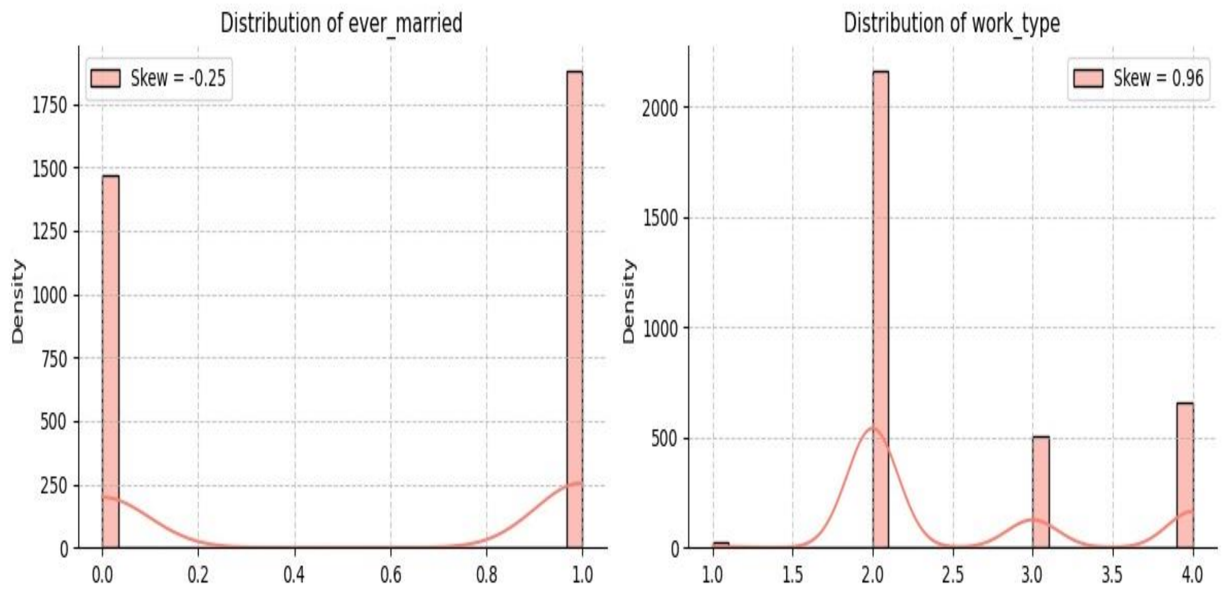


Figure 10: Histograms of Stroke data

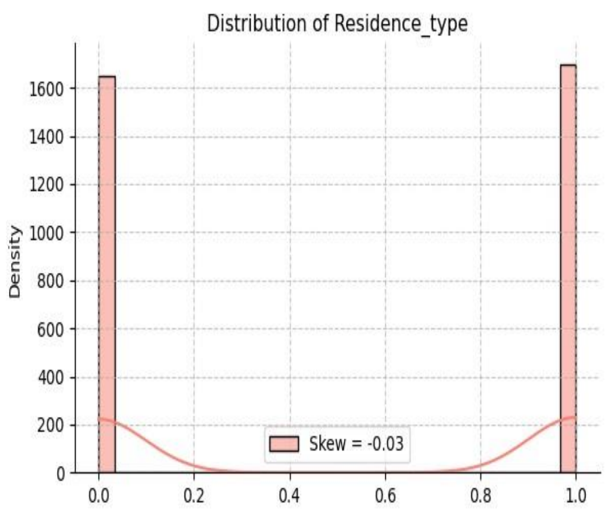


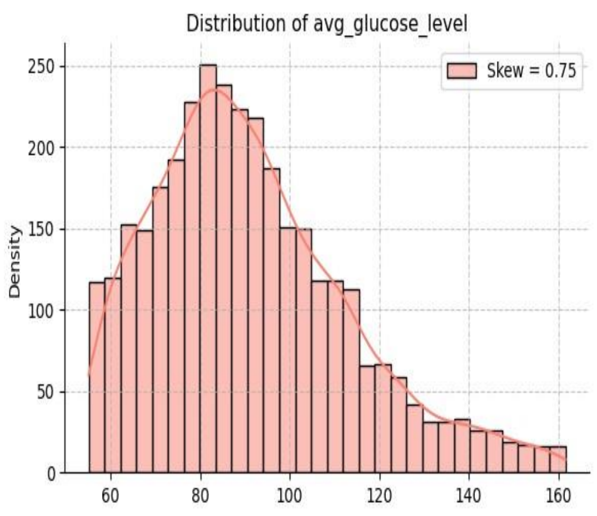
This code generates histograms to visualize the distribution of each numerical feature in the DataFrame

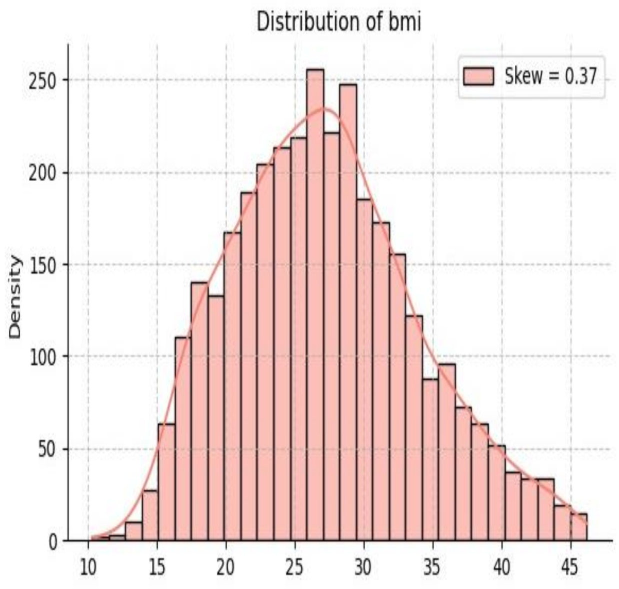
`df`. Here's a breakdown of what each part of the code does:

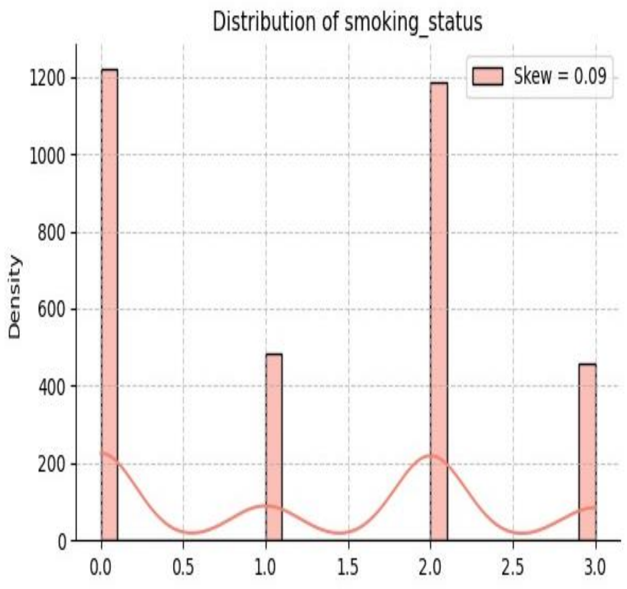
- `plt.figure(figsize=(12, 18))`: This line creates a new figure with a specific size (12 inches wide and

  18 inches tall) to contain the subplots.









- The `for` loop iterates each over of column in the DataFrame `df`, excluding the last column (assuming it's the target variable).
- `skewness = df[col].skew()`: This line calculates the skewness of the current column using the `skew()` method. Skewness measures the asymmetry of the distribution of values.
- `sns.histplot(df[col], kde=True, label=f"Skew = {skewness:.2f}", color='salmon', bins=30)`: This line creates a histogram of the values in the current column using Seaborn's `histplot()` function. It includes a kernel density estimate (kde), labels the skewness in the legend, sets the color to 'salmon', and divides the data into 30 bins.

- `plt.title(f"Distribution of {col}")`: This line sets the title of the subplot to indicate the column being visualized.
- `plt.xlabel('')`: This line removes the x-axis label.
- `plt.ylabel('Density')`: This line sets the y-axis label to 'Density'.
- `plt.legend(loc="best")`: This line adds a legend to the subplot, locating it at the best position. - `plt.grid(True, linestyle='--', alpha=0.7)`: This line adds gridlines to the subplot with a dashed line style and 70% transparency.
- `sns.despine()`: This line removes the top and right spines from the subplot.
- `plt.tight_layout()`: This line adjusts and sets the layout of the subplots to prevent the overlapping.

Finally, `plt.show()` displays the figure containing all the histograms.

This visualization allows you to quickly assess the distribution of each numerical feature in the dataset, including skewness, which can be more useful for understanding the data and making the decisions about data transformations if necessary.

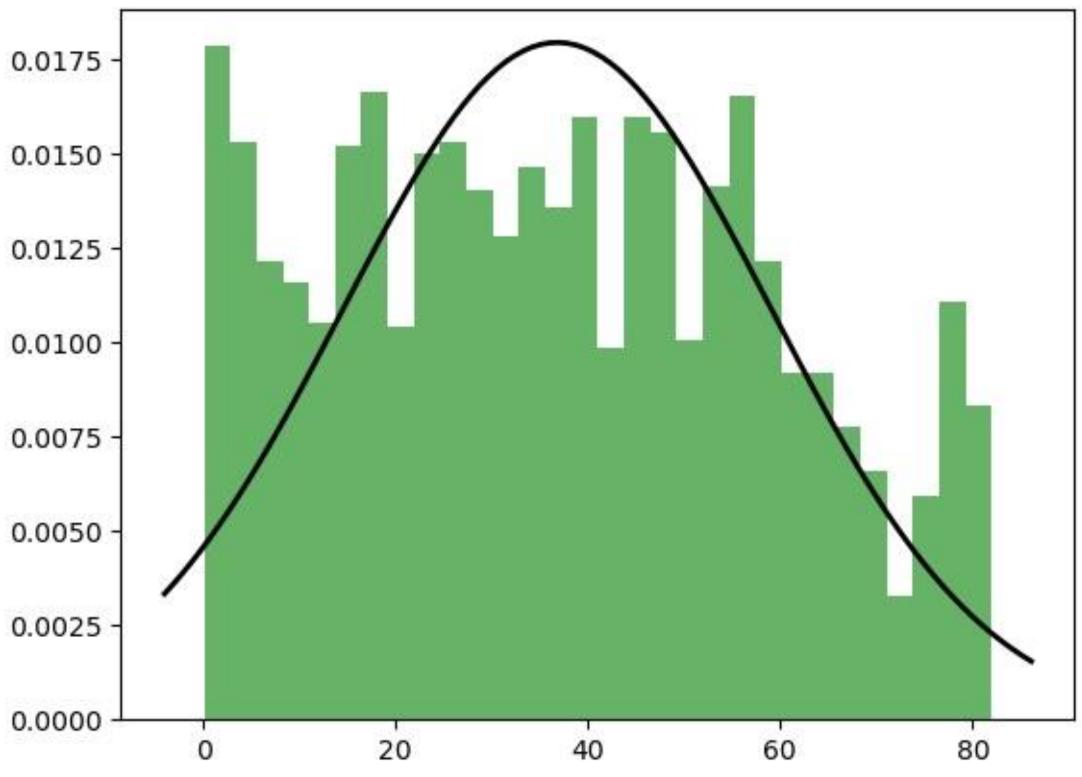


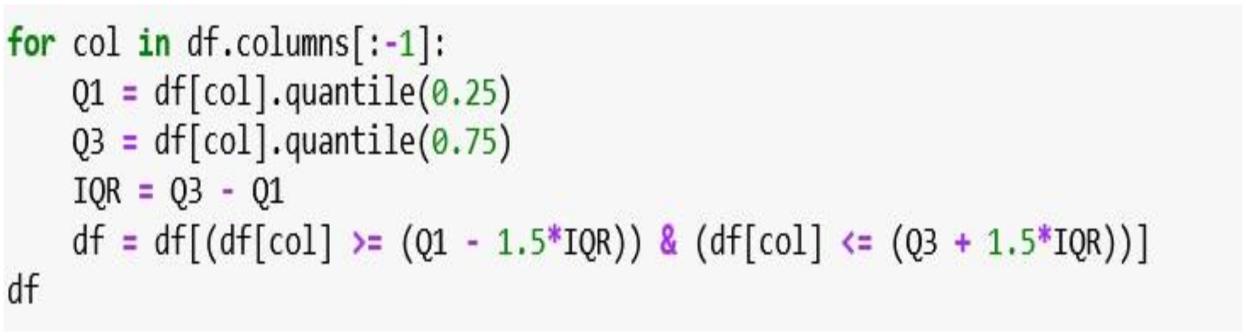
Figure 11: Distribution of numerical features

This code conducts the Shapiro-Wilk test for normality on each numerical feature in the DataFrame

`df`. Here's what each part of the code does:

- `import numpy as np`: Imports the Numpy library.
- `from scipy import stats`: Imports the `stats` module from SciPy, which includes statistical functions and tests.
- `from scipy.stats import shapiro`: Imports the `shapiro()` function specifically from the `stats` module to conduct the Shapiro-Wilk test for normality.
- The `for` loop iterates over each column in the DataFrame `df`, excluding the last column (assuming it's the target variable).
- `plt.hist(df[col], density=True, alpha=0.6, color='g', bins=30)`: This line creates a histogram of the values in the current column, setting the density parameter to `True`, the transparency to 0.6 (`alpha=0.6`), the color to green (`color='g'`), and dividing the data into 30 bins.
- `xmin, xmax = plt.xlim()`: This line determines the minimum and maximum values of the x-axis in the histogram.
- `x = np.linspace(xmin, xmax, 100)`: This line generates 100 evenly spaced points between `xmin` and `xmax` using NumPy's `linspace()` function.
- `p = stats.norm.pdf(x, np.mean(df[col]), np.std(df[col]))`: This line computes the probability density function (PDF) of a normal distribution with the mean and standard deviation of the current column, evaluated at the points `x`.
- `plt.plot(x, p, 'k', linewidth=2)`: This line plots the normal distribution curve (`p`) on top of the histogram, using a black line (`'k'`) with a linewidth of 2.
- `plt.show()`: This line displays the histogram and the normal distribution curve.
- `stat, p = shapiro(df[col])`: This line conducts the Shapiro-Wilk test for normality on the current column and returns the test statistic (`stat`) and the p-value (`p`).
- `print("Statistics = %.3f, p = %.3f" %(stat, p))`: This line prints the test statistic and the p-value.
- `alpha = 0.05`: This line sets the significance level (`alpha`) to 0.05.
- The `if` statement checks whether the p-value is greater than `alpha`.

- If `p > alpha`, it prints "Data looks Gaussian (fail to reject H0)," indicating that the data likely follows a Gaussian (normal) distribution.
- If `p <= alpha`, it prints "Data does not look Gaussian (reject H0)," indicating that the data likely does not follow a Gaussian distribution.

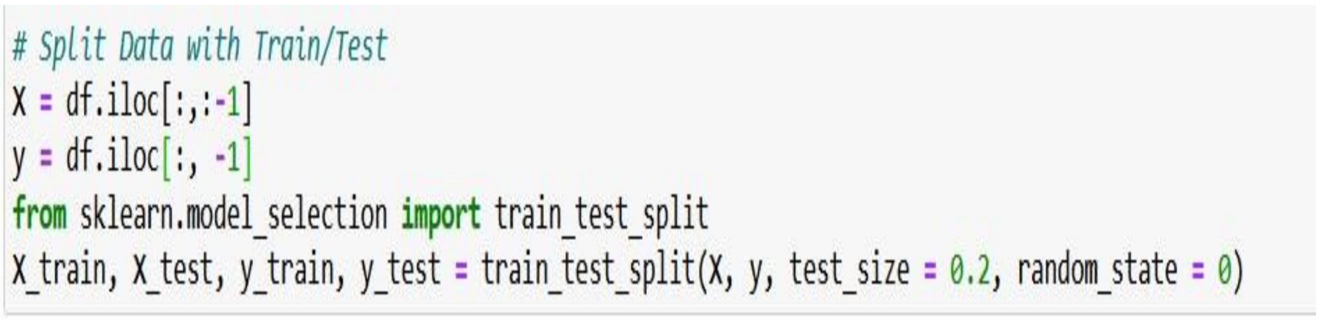


```
for col in df.columns[:-1]:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    df = df[(df[col] >= (Q1 - 1.5*IQR)) & (df[col] <= (Q3 + 1.5*IQR))]
df
```

- The `for` loop iterates over each column in the DataFrame `df`, excluding the last column (assuming it's the target variable).
- `Q1 = df[col].quantile(0.25)`: The first quartile deals and calculates this line (25th percentile) of the values in the colums.
- `Q3 = df[col].quantile(0.75)`: The third quartile calculates this line (75th percentile) of the values in the column.
- `IQR = Q3 - Q1`: This line calculates the interquartile range (IQR) of the values in the current column.
- `df = df[(df[col] >= (Q1 - 1.5*IQR)) & (df[col] <= (Q3 + 1.5*IQR))]`: This line filters the DataFrame `df` to keep only the rows where the values in the current column fall within a range defined by 1.5 times the IQR from the first and third quartiles. This range is commonly used to identify and remove outliers.

After applying outlier detection and removal to each numerical feature in the DataFrame `df`, the modified DataFrame `df` is returned, containing only the rows without outliers in any of the numerical features.

SPLITTING INTO TRAIN AND TEST:



```
# Split Data with Train/Test
X = df.iloc[:,:-1]
y = df.iloc[:, -1]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

By dividing the dataset into features (X) and the target variable (y), and then further dividing it into training and testing sets using the `train test split` function from scikit-learn, this gets the dataset ready for machine learning.

Here's what each part of the code does:

- `X = df.iloc[:,:-1]`: This line selects all the rows and all columns except the last one from the

DataFrame `df` to create the feature matrix `X`.

- `y = df.iloc[:, -1]`: This line selects the last column from the DataFrame `df` to create the target variable `y`.

`from sklearn.model_selection import train_test_split`: This line imports the `train_test_split` function from the `sklearn.model_selection` module, this divides datasets into arbitrary train and test subsets.

- `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)`: This line splits the dataset into training and testing sets. It takes the feature matrix `X` and target variable `y` as input, along with the `test_size` parameter specifying the proportion of the dataset to include in the test split (here set to 20%), and the `random_state` parameter to ensure reproducibility of the split.

Four sets of data will be available to you once this code is executed: `X_train} (training features), `X_test} (testing features), `y_train} (training target variable), and {y_test} (testing target variable).
Machine learning models are trained and assessed using these sets.

This code standardizes the features using `StandardScaler` from scikit-learn to ensure that features with different scales contribute equally to the model training process. Here's what each part of the code does:

`from sklearn.preprocessing import StandardScaler`: The `StandardScaler` class is imported from the `sklearn.preprocessing` module using this line. To standardize features, scale to unit variance and remove the mean using {StandardScaler}.

- `sc = StandardScaler()`: This line creates an instance of the `StandardScaler` class.
- `X_train = sc.fit_transform(X_train)`: This line applies the `fit_transform` method of the `StandardScaler` instance to standardize the training features (`X_train`). The `fit_transform` method fits the scaler to the training data and then transforms it. This ensures that the mean and standard deviation are computed from the training data and applied to both the training and testing data consistently.

- `X_test = sc.transform(X_test)`: This line applies the `transform` method of the `StandardScaler` instance to standardize the testing features (`X_test`). The `transform` method uses the mean and standard deviation computed during the fitting step (`sc.fit_transform(X_train)`) to transform the testing data.

After executing this code, the features in both the training (`X_train`) and testing (`X_test`) sets will be standardized, making them suitable for models that assume standard normally distributed data.

This code imports several machine learning models and evaluation metrics from scikit-learn. Here's what each part of the code does:

1. **Importing Machine Learning Models**:

- `LogisticRegression`: Importing the logistic regression classifier from scikit-learn, which is for binary classification in a linear model.
- `LinearRegression`: Importing the linear regression model from scikit-learn, which is used for regression tasks.
- `SVC`: Importing the support vector classifier (SVC) from scikit-learn, which is a popular model for classification tasks that can handle non-linear data.
- `DecisionTreeClassifier`: Importing the decision tree classifier from scikit-learn, which is a treebased model for classification tasks.
- `RandomForestClassifier`: Importing the random forest classifier from scikit-learn, which is an ensemble model consisting of multiple decision trees.

- `GradientBoostingClassifier`: Importing the gradient boosting classifier from scikit-learn, which is another ensemble model that builds trees in a sequential manner, each one correcting errors made by the previous tree.
- `AdaBoostClassifier`: Importing the AdaBoost classifier from scikit-learn, which is also an ensemble model that mixed up with multiple weak classifiers to enable or create a strong classifier.
- `KNeighborsClassifier`: Importing the k-nearest neighbors classifier from scikit-learn, which is a simple and effective classification algorithm based on proximity to neighboring data points.

2. **Importing Evaluation Metrics**:

- `accuracy_score`: Importing the function to calculate accuracy, which measures the proportion of correctly classified samples.

`precision_score`: Importing the function to calculate precisionevaluates the ratio of accurate positive forecasts to all positive forecasts.

`recall_score`: Importing the function to calculate recall, which calculates the ratio of accurate positive forecasts to all real positive occurrences.

- `f1_score`: Importing the function to calculate the F1 score, which is the harmonic mean of precision and recall and provides a balanced evaluation metric for binary classification tasks.

These imports set the framework for constructing and assessing machine learning models using various algorithms and assessing their performance using different evaluation metrics.

This function `accuracy` is designed to train multiple classification models using the provided training data (`X_train`, `y_train`) and then evaluate their performance on the provided testing data (`X_test`,

`y_test`). It returns the accuracy scores of each model.

Here's a breakdown of what the function does:

1. **Model Training**:

It initializes instances of many classification models, including AdaBoost Classifier (ABC), K-Nearest Neighbors Classifier (KNC), Gradient Boosting Classifier (GBC), Random Forest Classifier (RFC), Decision Tree Classifier (DTC), Support Vector Classifier (SVC), and Logistic Regression.

- It trains each model using the training data (`X_train`, `y_train`) using the `.fit()` method.

2. **Prediction**:

   - It makes predictions on the testing data (`X_test`) using each trained model (`reg`, `svc`, `dtc`, `rfc`,

`gbc`, `abc`, `knc`) using the `.predict()` method.

3. **Evaluation**:

   - It calculates the accuracy score for each model by comparing the predicted labels (`y_reg`, `y_svc`, `y_dtc`, `y_rfc`, `y_gbc`, `y_abc`, `y_knc`) with the true labels (`y_test`) using the `accuracy_score` function from scikit-learn.

4. **Return**:

- It returns a tuple containing the accuracy scores of all the models.

This function essentially provides a convenient way to evaluate the performance of multiple classification models on the provided testing data and compare their accuracy scores.

The code you provided creates an empty Data Frame named scores with columns representing different classifiers ("REG", "SVC", "DTC", "RFC", "GBC", "ABC", "KNC") and an index named "ACC". Then, it calculates the accuracy scores for each classifier using the accuracy function (which presumably evaluates multiple classifiers) and assigns these scores directly to the DataFrame. Finally, it displays the Data Frame containing the accuracy scores.

It looks like you've trained a logistic regression model with specific hyperparameters (`max_iter=50` and `class_weight="balanced"`) and evaluated its performance on both the training and testing sets.

Here's what each part of the code does:

- `reg = LogisticRegression(max_iter=50, class_weight="balanced")`: This line creates a logistic regression model with a maximum number of iterations set to 50 and with class weights balanced to handle class imbalance.
- `reg.fit(X_train, y_train)`: This line trains the logistic regression model on the training data (`X_train` and `y_train`).

- `print("REG Train Model Score :", reg.score(X_train, y_train))`: This line prints the accuracy score of the logistic regression model on the training data.
- `print("REG Test Model Score :", reg.score(X_test, y_test))`: This line prints the accuracy score of the logistic regression model on the testing data.

By printing these scores, you can assess how well the logistic regression model performs on both the training and testing sets. This helps you understand the model's generalization capability and whether it suffers from overfitting or underfitting.

Certainly! Let's break down the provided code step by step:

1. **Importing Libraries**:

- `import seaborn as sns`: This imports the Seaborn library, which is used for statistical data visualization.
- `import matplotlib.pyplot as plt`: This imports the Matplotlib library, which is a plotting library for

Python.

2. **Initializing Lists for Scores**:

- `rfc_train_scores = []`: This initializes an empty list to store the training scores of the Random Forest Classifier models.
- `rfc_test_scores = []`: This initializes an empty list to store the testing scores of the Random Forest Classifier models.

3. **Looping Over Different Values of `n_estimators`**:

- `for i in range(1, 10)`: This loop iterates over different values of the `n_estimators` parameter from

1 to 9.

4. **Training and Evaluating Random Forest Classifier Models**:

- `rfc = RandomForestClassifier(n_estimators=i * 50, max_depth=i + 1, max_features=i / 10, min_samples_split=i + 1)`: This line creates a Random Forest Classifier model with varying hyperparameters.
- `rfc.fit(X_train, y_train)`: The Random Forest Classifier model trains about this line on the training data (`X_train` and `y_train`).

`rfc_train_scores.append(rfc.score(X_train, y_train))`: This line calculates the training score (accuracy) of the trained Random Forest Classifier model on the training data and appends it to the `rfc_train_scores` list.

- `rfc_test_scores.append(rfc.score(X_test, y_test))`: This line calculates the testing score (accuracy) of the trained Random Forest Classifier model on the testing data and appends it to the `rfc_test_scores` list.

5. **Plotting**:

- `plt.figure(figsize=(10, 6))`: This line creates a new figure with a specified size for plotting.
- `sns.lineplot(x=range(1, 10), y=rfc_train_scores, marker='*', color='b', label='Training Score')`: This line plots the training scores against the different values of `n_estimators`. It uses asterisks as markers, blue color, and adds a label for the training scores.
- `sns.lineplot(x=range(1, 10), y=rfc_test_scores, marker='o', color='r', label='Testing Score')`: This line plots the testing scores against the different values of `n_estimators`. It uses circles as markers, red color, and adds a label for the testing scores.

6. **Adding Labels and Title**:

- `plt.xlabel('Number of Estimators (x50)')`: The X-axis were labelled for this line sets as "Number of Estimators (x50)".
- `plt.ylabel('Accuracy Score')`: The Y-axis were labelled for this line sets a "Accuracy Score".
- `plt.title('Random Forest Classifier Training and Testing Scores vs Number of Estimators')`: The plot of this line set as title.

7. **Showing the Plot**:

- `plt.legend()`: This line adds a legend to the plot to differentiate between the training and testing scores.

`plt.grid(True)`: This line enhances the plot's presentation by adding a grid.

- `plt.show()`: The plot were displayed by this line.

Overall, this code visualizes how the number of estimators affects the training and testing scores of the Random Forest Classifier model. It helps in understanding the trade-offs involved in choosing the number of estimators for a Random Forest Classifier.

It seems you've trained a Random Forest Classifier model with specific hyperparameters (`n_estimators=400`, `max_depth=9`, `max_features=0.8`, `min_samples_split=9`) and evaluated its performance on both the training and testing sets. Here's what each part of the code does:

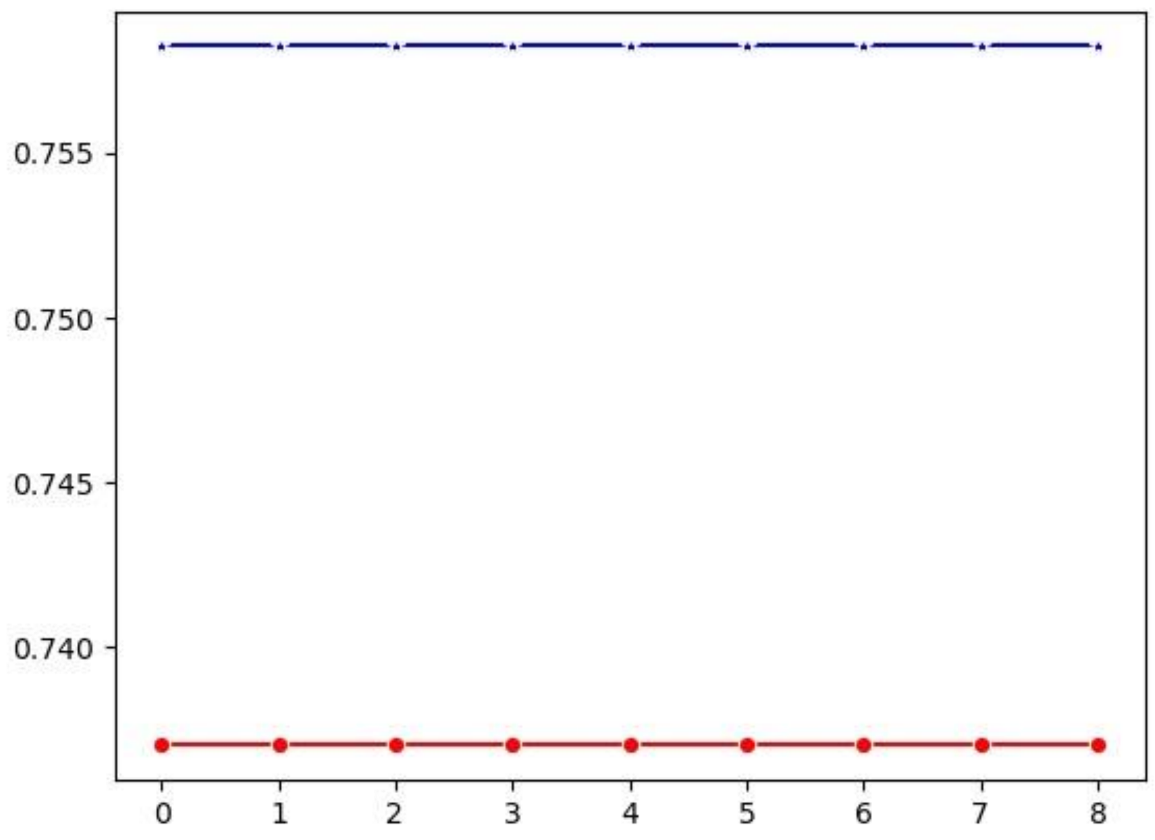
- `rfc = RandomForestClassifier(n_estimators=400, max_depth=9, max_features=0.8, min_samples_split=9)`: This line creates a Random Forest Classifier model with the specified hyperparameters.
- `n_estimators=400`: Number of trees in the forest.
- `max_depth=9`: Maximum depth of the trees.
- `max_features=0.8`: Maximum number of features to consider when splitting a node, specified as a fraction of the total number of features.
- `min_samples_split=9`: Minimum number of samples required to split an internal node.
- `rfc.fit(X_train, y_train)`: This line trains the Random Forest Classifier model on the training data

(`X_train` and `y_train`).

- `print("RFC Train Model Score :", rfc.score(X_train, y_train))`: This line prints the accuracy score of the Random Forest Classifier model on the training data.
- `print("RFC Test Model Score :", rfc.score(X_test, y_test))`: This line prints the accuracy score of the

Random Forest Classifier model on the testing data.

By printing these scores, you can assess how well the Random Forest Classifier model performs on both the training and testing sets. This helps you understand the model's generalization capability and whether it suffers from overfitting or underfitting.



Certainly! Let's break down the code step by step:

1. **Importing Libraries**:

- `import seaborn as sns`: This imports the Seaborn library, a tool for visualizing statistical data.

- `import matplotlib.pyplot as plt`: This imports the Matplotlib library, which is a plotting library for

Python.

2. **Initializing Lists for Scores**:

- `knc_train_scores = []`: This initialize an empty list to pack the training scores of the k-nearest neighbors (KNN) classifier models.
- `knc_test_scores = []`: This initializes an empty list to store the testing scores of the KNN classifier models.

3. **Looping Over Different Values of `n_neighbors`**:

- `for i in range(1, 10)`: This loop iterates over different values of the `n_neighbors` parameter from 1 to 9.

4. **Training and Evaluating KNN Models**:

- `knc = KNeighborsClassifier(n_neighbors=i)`: This line creates a KNeighborsClassifier model with the current value of `n_neighbors`.
- `knc.fit(X_train, y_train)`: This line trains the KNN model on the training data (`X_train` and `y_train`).
- `knc_train_scores.append(knc.score(X_train, y_train))`: This line calculates the training score

(accuracy) of the trained KNN model on the training data and appends it to the `knc_train_scores` list.

- `knc_test_scores.append(knc.score(X_test, y_test))`: This line calculates the testing score

(accuracy) of the trained KNN model on the testing data and appends it to the `knc_test_scores` list.

5. **Plotting**:

- `plt.figure(figsize=(10, 6))`: This line creates a new figure with a specified size for plotting.
- `sns.lineplot(x=range(1, 10), y=knc_train_scores, marker='*', color='b', label='Training Score')`: This line plots the training scores against the different values of `n_neighbors`. It uses asterisks as markers, blue color, and adds a label for the training scores.
- `sns.lineplot(x=range(1, 10), y=knc_test_scores, marker='o', color='r', label='Testing Score')`: This line plots the testing scores against the different values of `n_neighbors`. It uses circles as markers, red color, and adds a label for the testing scores.

6. **Adding Labels and Title**:

`plt.xlabel('Number of Neighbors')`: This line establishes the x-axis label as "Number of Neighbors".

`plt.ylabel('Accuracy Score')`: This line establishes the x-axis label as"Accuracy Score".

- `plt.title('K-Nearest Neighbors Classifier Training and Testing Scores vs Number of Neighbors')`:

The plot of the title were setted by this line.

7. **Showing the Plot**:

`plt.legend()`: This sentence gives the story a legend to differentiate between the training and testing scores.

`plt.grid(True)`: This line enhances the plot's presentation by adding a grid.

- `plt.show()`: This line displays the plot.

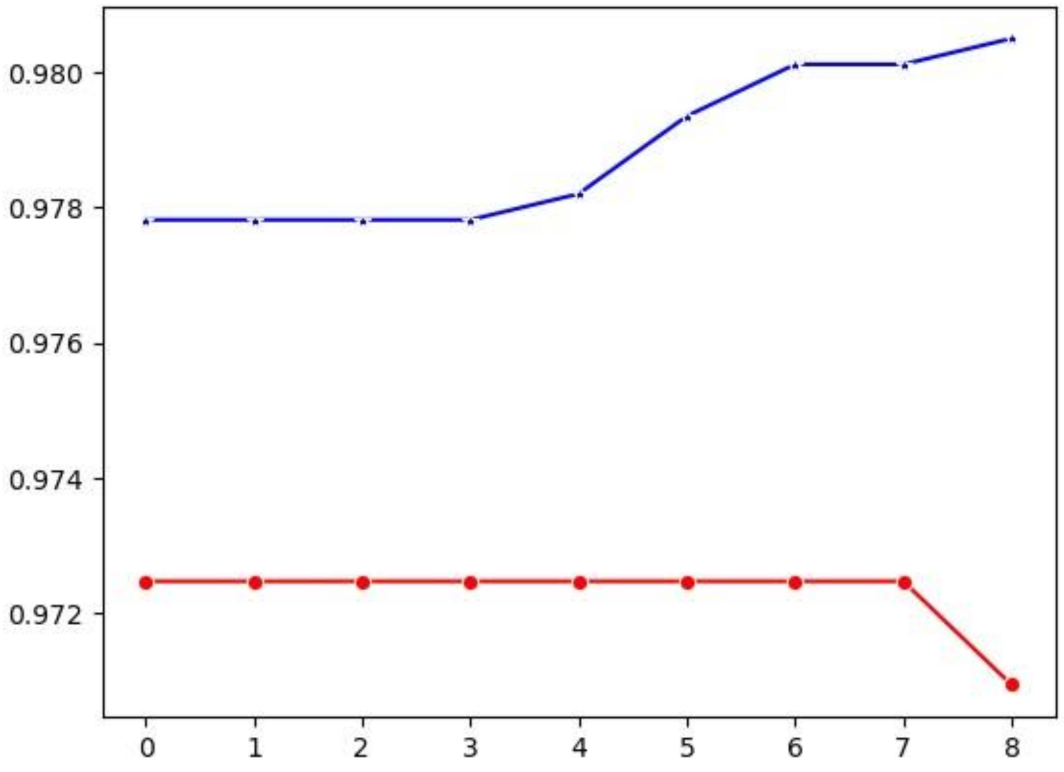


Figure 12: Plot of stroke data

Overall, this code trains multiple KNN models with different numbers of neighbors, evaluates their performance on both training and testing data, and visualizes the results using a line plot. This visualization helps in understanding how the choice of the number of neighbors affects the performance of the KNN classifier.

In this code, you're training a k-nearest neighbors (KNN) classifier with a specific number of neighbors (`n_neighbors = 50`) and evaluating its performance on both the training and testing sets. Here's what each part of the code does:

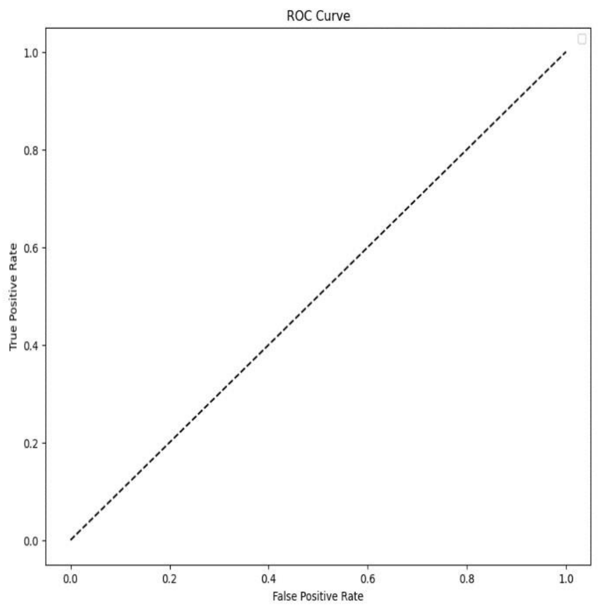
`knc = KNeighborsClassifier(n_neighbors=50)`: This line creates a KNeighborsClassifier model with `n_neighbors` set to 50. In KNN, the `n_neighbors` parameter specifies the number of neighbors to consider when making predictions.

- `knc.fit(X_train, y_train)`: This line trains the KNN model using the training data (`X_train` and

`y_train`), where `X_train` contains the features and `y_train` contains the corresponding target labels.

- `print("KNC Train Model Score :", knc.score(X_train, y_train))`: This line prints the accuracy score of the trained KNN model on the training data. The `score` method of the KNN classifier calculates the mean accuracy on the given training data and labels.
- `print("KNC Test Model Score :", knc.score(X_test, y_test))`: This line prints the accuracy score of the trained KNN model on the testing data. Similar to the previous line, the `score` method is used to calculate the mean accuracy on the given testing data and labels.

By printing these scores, you can assess how well the KNN model performs on both the training and testing sets. This helps you understand the model's ability to generalize to unseen data and whether it suffers from overfitting or underfitting.



This code segment is performing several analyses and visualizations for evaluating the classifiers. Let's break it down:

1. **Importing Necessary Libraries**:

- `from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score, precision_recall_curve, average_precision_score`: This imports various evaluation metrics such as confusion matrix, ROC curve, ROC AUC score, precision-recall curve, and average precision score from scikit-learn.   - `from sklearn.base import is_classifier`: This imports the function `is_classifier` from scikit-learn, which checks if an object is a classifier.

2. **Filtering Classifier List**:

  - `classifier_list = [classifier for classifier in scores if is_classifier(classifier)]`: This line creates a list of classifiers from the `scores` DataFrame. It filters out non-classifier objects using the `is_classifier` function.

3. **ROC Curve**:

This segment plots the ROC curve for each classifier in `classifier_list`. Using the `roc_curve} function, it determines the true positive rate (TPR) and false positive rate (FPR) and plots them. A dashed diagonal line represents a random classifier.

- `roc_curve` computes the ROC curve.

- `roc_auc_score` computes the area under the ROC curve (AUC).

- The plot is labeled with each classifier's name and its corresponding AUC.

4. **Precision-Recall Curve**:

- This segment plots the precision-recall curve for each classifier in `classifier_list`. It calculates precision and recall using the `precision_recall_curve` function and plots them.
- `average_precision_score` computes the average precision (AP).

- The plot is labeled with each classifier's name and its corresponding AP.

5. **Feature Importance**:

- This segment plots the top 10 feature importances for either the `RandomForestClassifier` or the `GradientBoostingClassifier`, depending on which one was trained (`rfc` or `gbc`).
- If `rfc` is an instance of `RandomForestClassifier`, it plots the feature importances using `sns.barplot`.
- If `gbc` is an instance of `GradientBoostingClassifier`, it plots the feature importances similarly.

- The features are sorted based on their importance, and the top 10 features are plotted.

These analyses and visualizations give information about the actions and conduct of the classifiers, helping in model selection, tuning, and feature importance assessment.

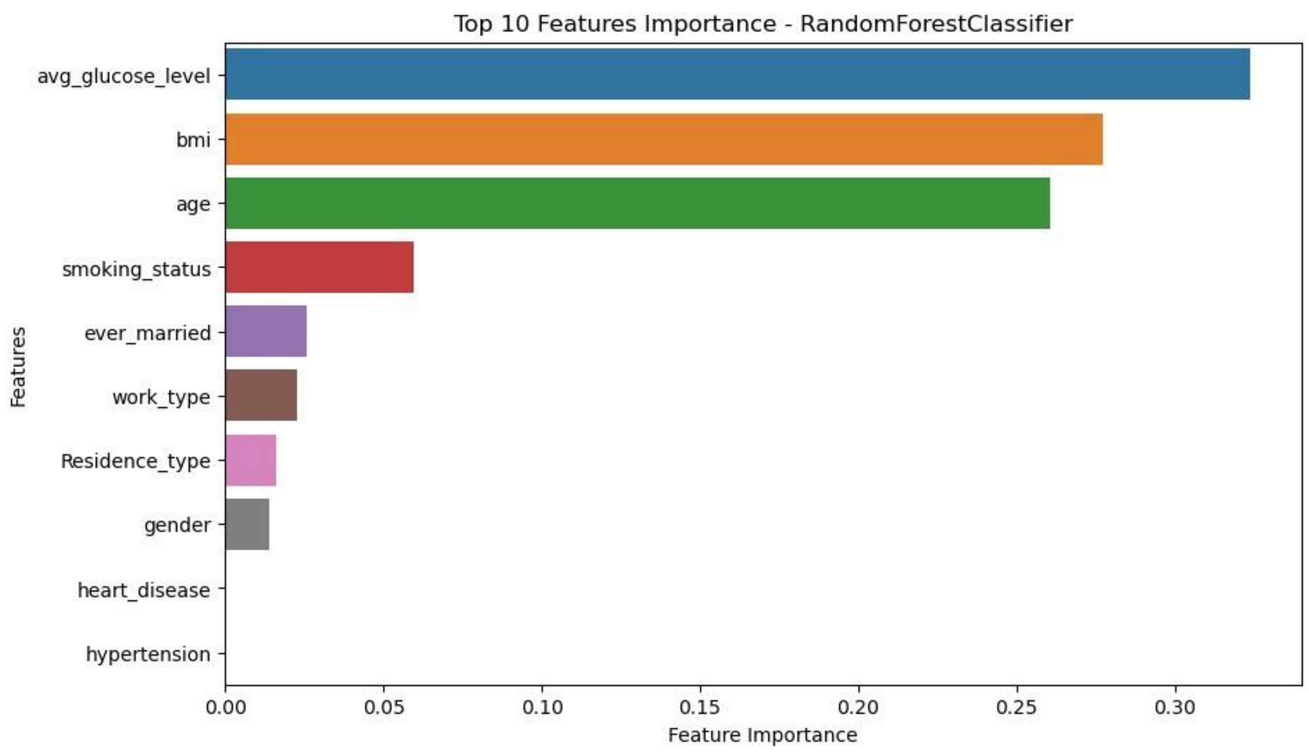


Figure 13: Random Forest Classifier

CHAPTER 5

SYSTEM REQUIREMENTS

5.1 HARDWARE REQUIREMENTS:

- System : Pentium IV 2.4 GHz
- Hard Disk : 200 GB
- Mouse : Logitech.
- Keyboard : 110 keys enhanced
- Ram : 4GB

5.2 SOFTWARE REQUIREMENTS:

- O/S : Windows 7.
- Language : Python
- Front End : Jupiter

5.3 SOFTWARE DESCRIPTION:

5.3.1 Python

Python is one of the few languages that can claim to be both simple and robust. You'll be pleasantly pleased at how easy it is to focus on the solution to the problem rather than the syntax and structure of the programming language. The official introduction to Python is that it is a simple yet powerful programming language. It employs efficient high-level data structures and a straightforward but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it a great language for scripting and quick application development in a variety of fields across most platforms. In the following part, I will go into greater depth about the majority of these features.

5.3.2 Features of Python

- **Simple**

Python is an easy, simple language. Reading a decent Python program feels similar to reading English, albeit extremely rigorous English! Python's pseudo-code nature is one of its most notable strengths. It allows you to focus on the solution rather than the language.

- **Easy to Learn**

As you can see, Python is fairly simple to learn. As previously said, Python's syntax is incredibly easy.

- **Free and Open Source**

Python is an example of a *FLOSS* (Free/Libré and Open Source Software). In simple terms, you can freely distribute copies of this software, read its source code, make changes to it, and use pieces of it in new free programs. FLOSS is based on the concept of a community which shares knowledge. This is one of the reasons why Python is so good - it has been created and is constantly improved by a community who just want to see a better Python.

- **High-level Language**

When you write programs in Python, you never need to bother about the low-level details such as managing the memory used by your program, etc.

- **Portable**

Python is open-source, which means it may be customized to run on a variety of platforms. If you avoid using system-specific capabilities, your Python programs will run effortlessly on all of these systems. Python is compatible with several operating systems, including GNU/Linux, Windows, FreeBSD, macOS, Solaris, OS/2, Amiga, and AROS. It is also compatible with devices such as PlayStation and PocketPC. You may also use frameworks like Kivy to create games on iPhone, iPad, and Android devices, as well as your computer.

- **Interpreted**

A program written in a compiled language, such as C or C++, is transformed from the source language (C or C++) to a language that your computer understands (binary code, i.e. 0s and 1s) by a compiler with various flags and parameters. When you run the program, the linker/loader software copies it from the hard drive to memory and begins running it. Python, on the other hand, does not require compilation to binary. You simply run the application straight from the source code. Internally, Python turns the original code into an intermediate form known as byte codes, which are then translated into your computer's local language and executed.

- **Object Oriented**

Python is versatile, allowing for both procedural and object-oriented programming techniques. In procedural languages, the emphasis is on functions or processes, which are code parts that can be reused. Object-oriented languages, on the other hand, are built around objects that represent data and activity. Python provides an effective and user-friendly implementation of OOP, particularly when contrasted to larger languages like C++ and Java.

- **Extensive Libraries**

The Python Standard Library is extensive and supports a wide range of tasks, including regular expressions, documentation generation, unit testing, threading, database management, web browsers, CGI, FTP, email, XML, XML-RPC, HTML, audio files, cryptography, and graphical user interfaces, among other system-specific functions. All of these features are available wherever Python is deployed, demonstrating Python's "Batteries Included" approach. In addition to the standard library, the Python Package Index contains additional high-quality packages.

CHAPTER 6

CONCLUSION

In a nutshell up, this study shows that sophisticated machine learning methods, like the convolution Neural Network (CNN) with Random Forest classifier, can accurately predict the likelihood of strokes with 94% and 94.9% accuracy rates, respectively. These models offer precise risk assessments and tailored intervention methods by examining a variety of demographic and health-related variables, such as age, smoking status, average glucose level, history of heart disease, hypertension status, BMI.

Future Enhancement

In future, it will hybrid the two different machine learning or to combine the two different deep learning algorithm or machine with deep learning algorithm. In future, we can implement the web application.

Publication Details

Name of the Journal: 14TH INTERNATIONAL CONFERENCE ON SCIENCE AND INNOVATIVE ENGINEERING -14 ICSIE 2024

Paper ID: ICSIE240954

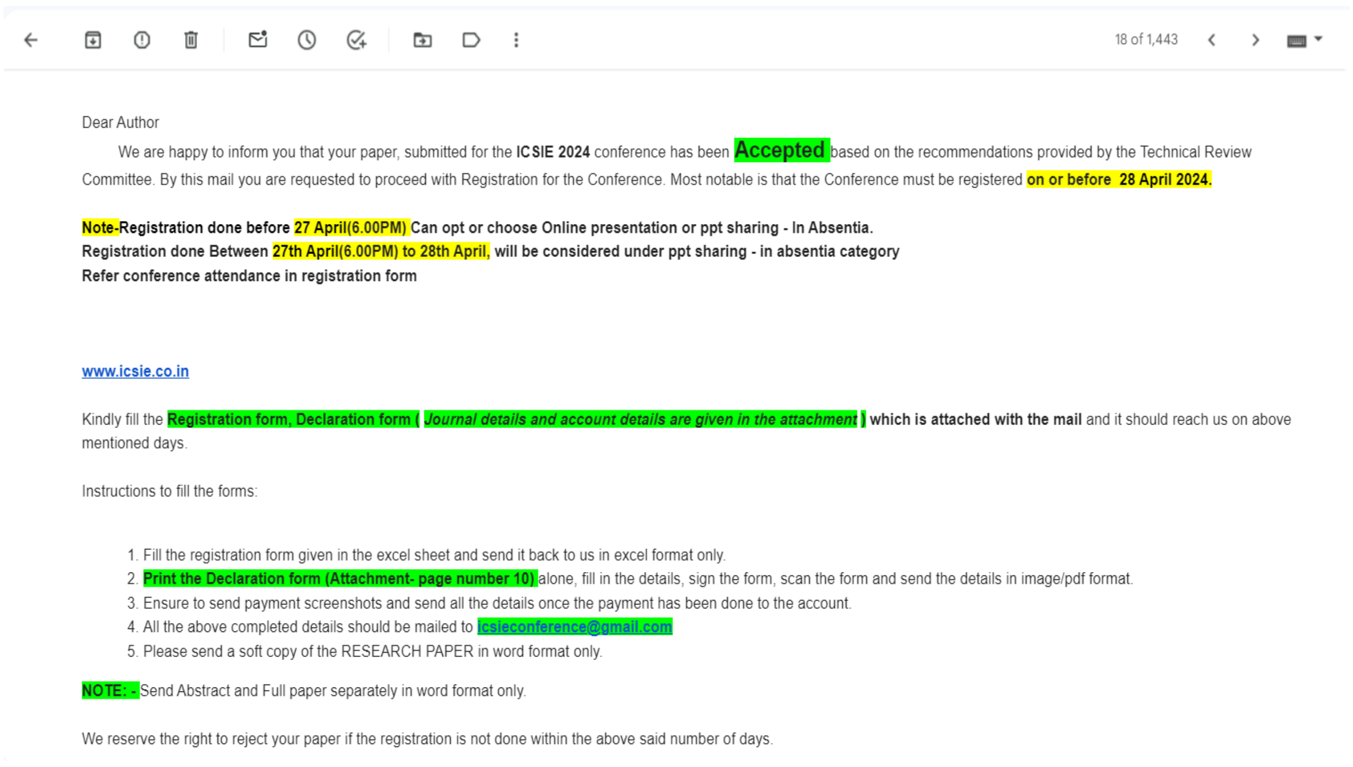
Manuscript title: Predicting Stroke Risk Using Hybrid Deep Transfer Learning Models

Authors: Abburi Naveen Varma, Ajay Kumar Bandhan, Morrigadudhula Abhinay, Mattapalli Veerasai, Bavanari AnilKumar, Somishetty Sanjay Varma, Chinda Harshavardhan Raju.

Status of the paper: ACCEPTED

Volume: Volume 08

Issue: Issue 27 April – 2024



Dear Author

We are happy to inform you that your paper, submitted for the **ICSIE 2024** conference has been **Accepted** based on the recommendations provided by the Technical Review Committee. By this mail you are requested to proceed with Registration for the Conference. Most notable is that the Conference must be registered **on or before 28 April 2024.**

Note-Registration done before 27 April(6.00PM) Can opt or choose Online presentation or ppt sharing - In Absentia.
Registration done Between 27th April(6.00PM) to 28th April, will be considered under ppt sharing - in absentia category
Refer conference attendance in registration form

www.icsie.co.in

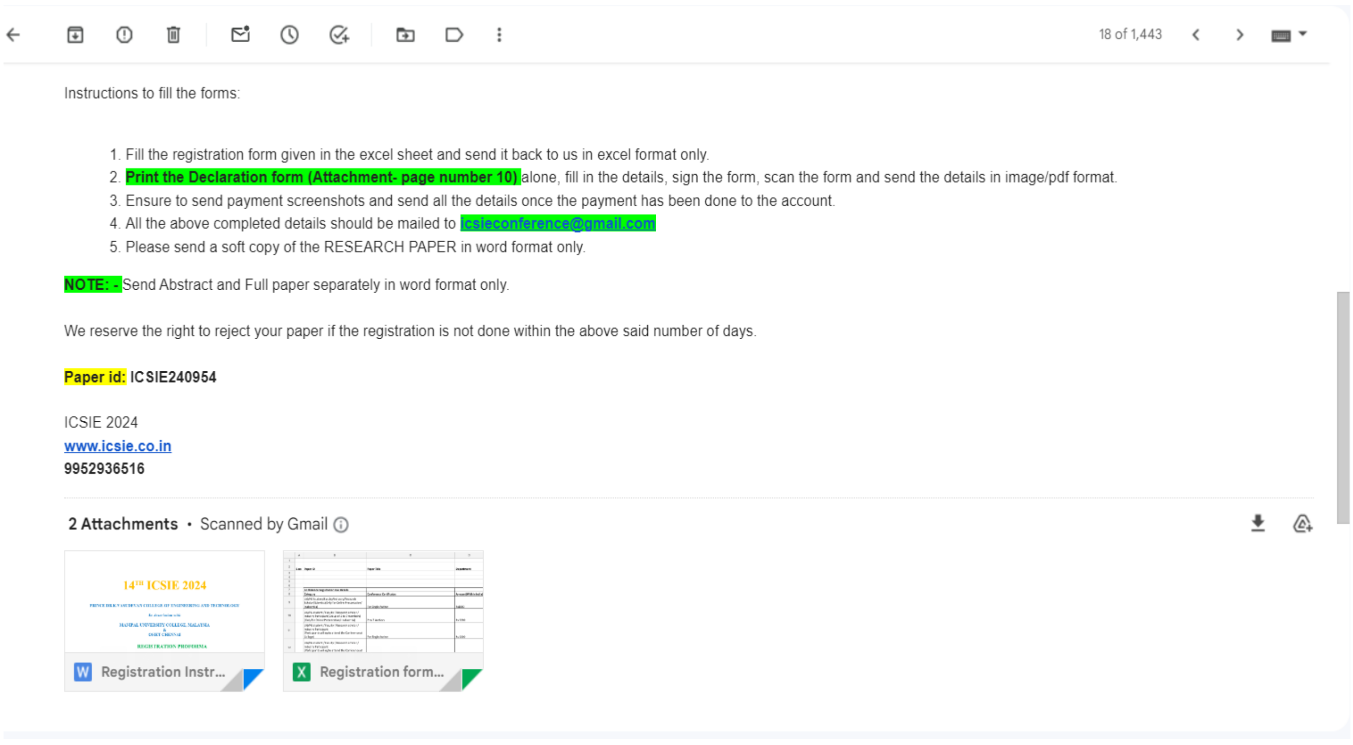
Kindly fill the **Registration form, Declaration form (** ***Journal details and account details are given in the attachment*** **) which is attached with the mail** and it should reach us on above mentioned days.

Instructions to fill the forms:

1. Fill the registration form given in the excel sheet and send it back to us in excel format only.
2. **Print the Declaration form (Attachment- page number 10)** alone, fill in the details, sign the form, scan the form and send the details in image/pdf format.
3. Ensure to send payment screenshots and send all the details once the payment has been done to the account.
4. All the above completed details should be mailed to icsieconference@gmail.com
5. Please send a soft copy of the RESEARCH PAPER in word format only.

NOTE: - Send Abstract and Full paper separately in word format only.

We reserve the right to reject your paper if the registration is not done within the above said number of days.



Instructions to fill the forms:

1. Fill the registration form given in the excel sheet and send it back to us in excel format only.
2. **Print the Declaration form (Attachment- page number 10)** alone, fill in the details, sign the form, scan the form and send the details in image/pdf format.
3. Ensure to send payment screenshots and send all the details once the payment has been done to the account.
4. All the above completed details should be mailed to icsieconference@gmail.com
5. Please send a soft copy of the RESEARCH PAPER in word format only.

NOTE: - Send Abstract and Full paper separately in word format only.

We reserve the right to reject your paper if the registration is not done within the above said number of days.

Paper id: ICSIE240954

ICSIE 2024
www.icsie.co.in
9952936516

2 Attachments • Scanned by Gmail

REFERENCES

[1] Cui, L., Fan, Z., Yang, Y., Li, R., Wang, D., Feng, Y., … & Fan, Y. (2022). Deep learning in ischemic stroke imaging analysis: a comprehensive review. BioMed Research International, 2022, 1-15.

[2] Wei, Z., Li, M., & Fan, H. (2022). Hybrid deep learning model for the risk prediction of cognitive impairment in stroke patients.

[3] Chantamit-o-pas, P. and Goyal, M. (2017). Prediction of stroke using deep learning model. Neural Information Processing, 774-781.

[4] S. V, R. and R, G. (2023). Hybrid deep transfer learning framework for stroke risk prediction. The International Conference on Scientific Innovations in Science, Technology, and Management.

[5] Rao, B. N., Mohanty, S., Sen, K., Acharya, U. R., Cheong, K. H., & Sabut, S. (2022). Deep transfer learning for automatic prediction of hemorrhagic stroke on ct images. Computational and Mathematical Methods in Medicine, 2022, 1-10.

[6] Cheon, S., Kim, J., & Lim, J. (2019). The use of deep learning to predict stroke patient mortality. International Journal of Environmental Research and Public Health, 16(11), 1876.

[7] AlArfaj, A. A., Mahmoud, H. A. H., & Hafez, A. M. (2022). A deep learning model for stroke patients' motor function prediction. Applied Bionics and Biomechanics, 2022, 1-9.

[8] Liu, Y., Yu, Y., Ouyang, J., Jiang, B., Yang, G., Ostmeier, S., … & Zaharchuk, G. (2023). Functional outcome prediction in acute ischemic stroke using a fused imaging and clinical deep learning model. Stroke, 54(9), 2316-2327.

[9] Su, S., Li, L., Wang, Y., & Li, Y. (2023). Stroke risk prediction by color Doppler ultrasound of carotid artery-based deep learning using Inception V3 and VGG-16. *Frontiers in Neurology, 14*.

[10] Lee, J.H., Kwon, J., Lee, M.S., Cho, Y., Oh, I., Park, J.J., & Jeon, K. (2022). Prediction of atrial fibrillation in patients with embolic stroke with undetermined source using electrocardiogram deep learning algorithm and clinical risk factors. *European Heart Journal*.

[11] Kim, D., Choi, K., Kim, J., Hong, J.W., Choi, S., Park, M., & Cho, K. (2023). Deep learning-based personalised outcome prediction after acute ischaemic stroke. *Journal of Neurology, Neurosurgery, and Psychiatry, 94*, 369 - 378.

[12] Matsui, H., Yamana, H., Fushimi, K., & Yasunaga, H. (2021). Development of Deep Learning Models for Predicting In-Hospital Mortality Using an Administrative Claims Database: Retrospective Cohort Study. *JMIR Medical Informatics, 10*.

[13] Sharma, N., Mishra, M.K., Chadha, J.S., & Lalwani, P. (2021). Heart Stroke Risk Analysis: A Deep Learning Approach. *2021 10th IEEE International Conference on Communication Systems and Network Technologies (CSNT)*, 543-598.

[14] Rehman, A. (2023). Brain Stroke Prediction through Deep Learning Techniques with ADASYN Strategy. *2023 16th International Conference on Developments in eSystems Engineering (DeSE)*, 679-684.

[15] Prof.SomashekharB, M., & Gangatkar, S.N. (2023). DEEP LEARNING APPROACH FOR CHOLESTEROL DETECTION AND STROKE PREDICTION.

[16] Lim, C.C., Chong, C., Tan, G., Tan, C.S., Cheung, C.Y., Wong, T.Y., Cheng, C., & Sabanayagam, C. (2023). A deep learning system for retinal vessel calibre improves cardiovascular risk prediction in Asians with chronic kidney disease. *Clinical Kidney Journal, 16*, 2693 - 2702.

[17] Chantamit-o-pas, P., & Goyal, M.L. (2017). Prediction of Stroke Using Deep Learning Model. *International Conference on Neural Information Processing*.

[18] Lee, S., Lee, S.H., Choi, H., Lee, J., Jeong, Y.W., Kang, D.R., & Sung, K.C. (2022). Deep Learning Improves Prediction of Cardiovascular Disease-Related Mortality and Admission in Patients with Hypertension: Analysis of the Korean National Health Information Database. *Journal of Clinical Medicine, 11*.

[19] Li, Y., Salimi-Khorshidi, G., Rao, S., Canoy, D., Hassaine, A., Lukasiewicz, T., Rahimi, K., & Mamouei, M. (2022). Validation of risk prediction models applied to longitudinal electronic health record data for the prediction of major cardiovascular events in the presence of data shifts. *European Heart Journal. Digital Health, 3*, 535 - 547.

[20] Liang, H., Liang, D., Tao, L., Zhang, X., Li, X., Zheng, D., Chen, Y., Yi, M., & Tang, F. (2021). Predict the Risk of Dyslipidemia Using Electronic Health Records: Apply Deep Neural Networks for Survival Data (Preprint).