DEPI Final Project

Streamlit

Stroke prediction

By:
Rehab Yehia
Nada Mohamed
Mahmoud Ahmed|
Weam Taie
Kamal Khaled
Mohamed Fouda

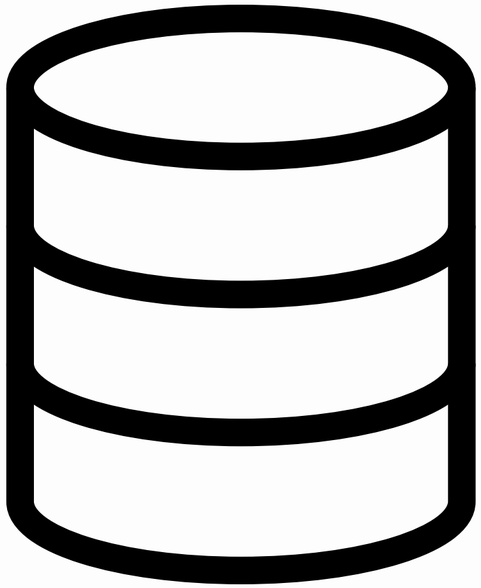
Overview

About dataset

•**Dataset Source**: Kaggle Stroke Prediction Dataset

Purpose: Analyze various factors contributing to stroke risk.

Goal: The primary objective of this project is to develop and evaluate machine learning models to accurately predict stroke occurrences in patients.



Given the life-threatening nature of strokes, the focus is on optimizing the model to minimize false negatives, where a stroke goes undetected.

Reducing these errors is critical:

- to ensuring timely medical intervention
- potentially saving lives
- preventing health complications.

Metadata

Total Records: 5,110 entries

	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

Metadata

Column	Description
ID	A unique identifier for each patient
Gender	The patient's gender, which could be "Male," "Female," or "Other,"
Age	The age of the patient in years, a critical factor in assessing stroke risk
Hypertension	Indicates whether the patient has hypertension (0 = No, 1 = Yes)
Heart Disease	Shows whether the patient has a history of heart disease (0 = No, 1 = Yes)
Ever Married	Denotes the patient's marital status as "Yes" or "No,"
Work Type	The type of employment the patient engages in, categorized as "children," "Govt_job," "Never_worked," "Private," or "Self-employed."

Metadata

Column	Description
Residence Type	The patient’s living environment, classified as "Urban" or "Rural,"
Avg Glucose Level	The patient's average blood glucose level
BMI	Body Mass Index, representing the patient’s weight relative to height
Smoking Status	Classifies the patient as "formerly smoked," "never smoked," "smokes," or "Unknown,"
Stroke	The target variable, where 1 indicates the patient has experienced a stroke and 0 indicates no stroke

DATA PREPRATION

As it is so mature Data ,We Only need to:

01 Handle Missing Values: BMI had missing values which were addressed.

02 Data Cleaning: Removing duplicates and invalid entries.

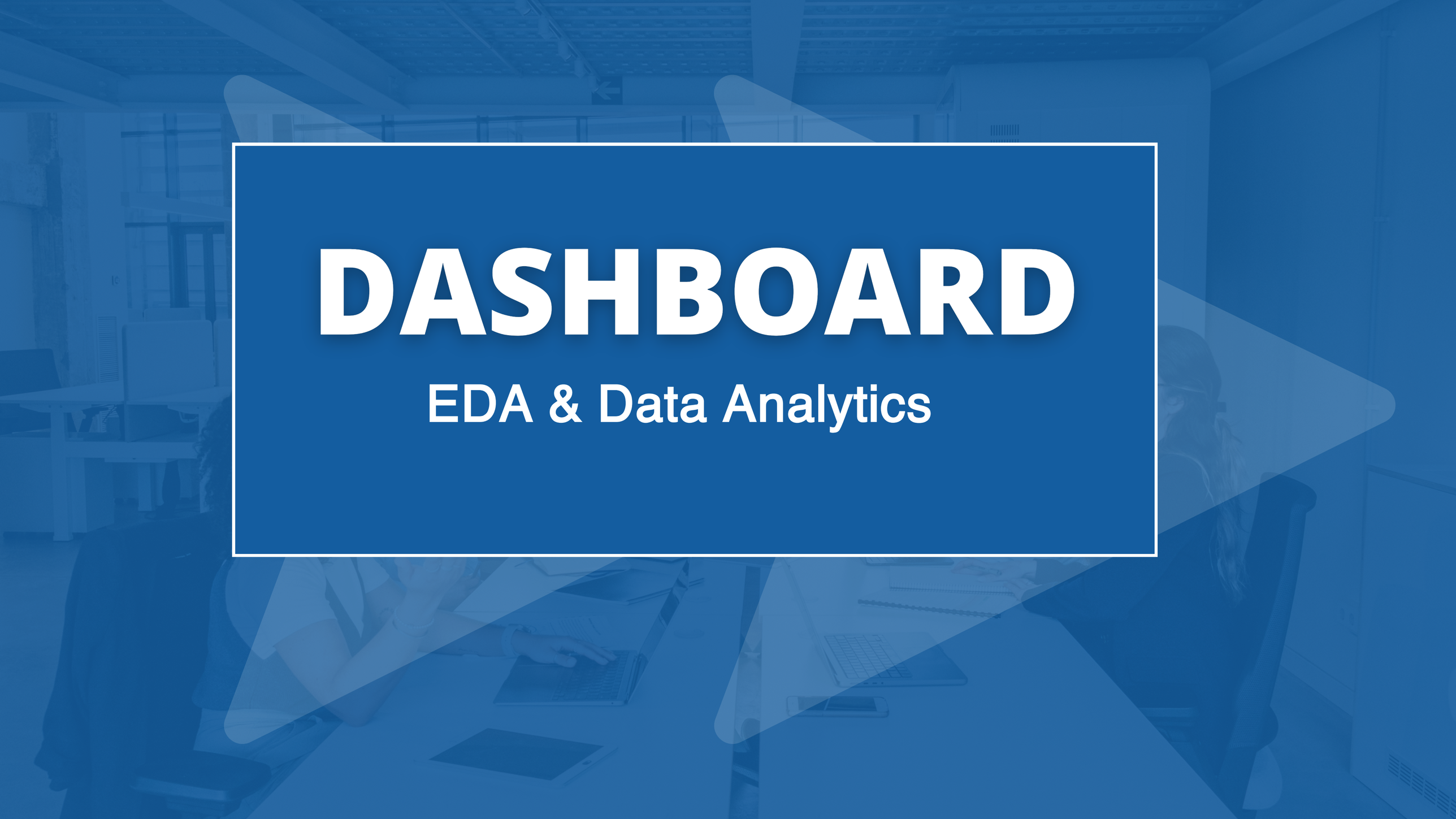
Problem 1: Check for Duplicates

Duplicate Records: The dataset was scanned for duplicate patient entries based on unique identifiers.
o Solution: No duplicate records were found, ensuring data accuracy.

Problem 2: Check for Missing Values

- BMI and Smoking Status: Missing values were found in the "bmi" and "smoking_status" columns, which are crucial factors in predicting stroke risk.

- Solution: Missing values in the "bmi" column were imputed using the median value of BMI to avoid skewing the data, while "Unknown" values in the "smoking_status" column were left as is but carefully considered in the analysis.



DASHBOARD

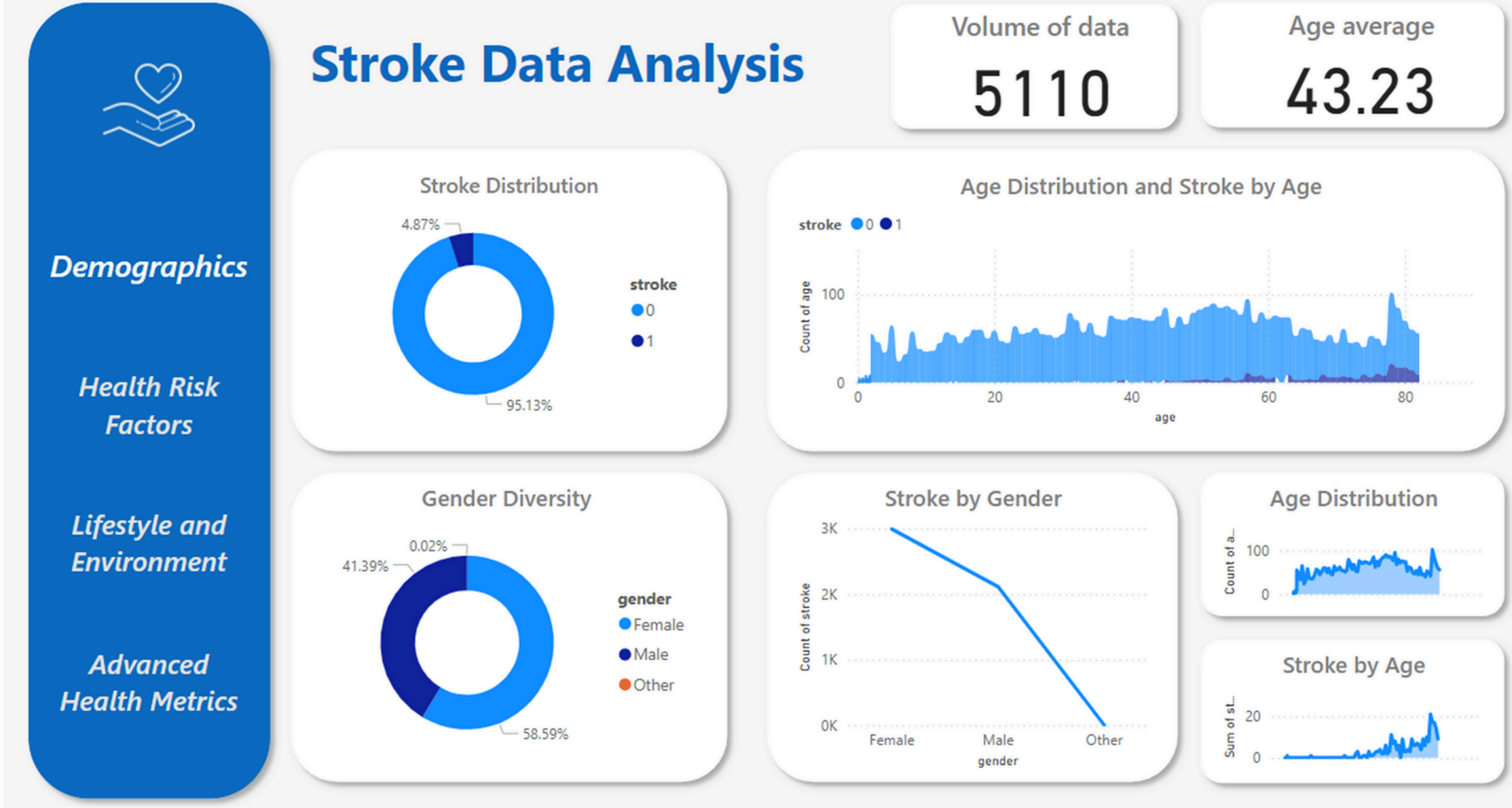
EDA & Data Analytics

Objectives

To achieve the overarching goal,
the study will focus on several key objectives:

- Analyze demographic trends
- Heath risk factors
- Evaluate lifestyle factors
- Examine the health metrics

Demographics



Insights

•Average Age: The chart shows an average age of 43.23, derived from a data sample of 5110 people.

•Stroke Distribution Pie Chart: A pie chart highlights that 95.13% of the population has never experienced a stroke, while only 4.87% have. This shows that strokes are relatively uncommon in the dataset.

•Age Distribution & Stroke by Age (Bar Graph): This dual-axis bar graph demonstrates how stroke rates increase with age, showing that the majority of stroke cases occur in older populations.

•Stroke by Gender (Bar Chart): This bar chart shows stroke counts by gender, revealing a slightly higher stroke count in males compared to females.

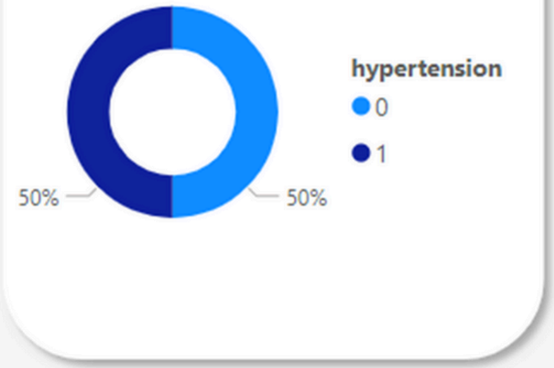
Health Risk Factors

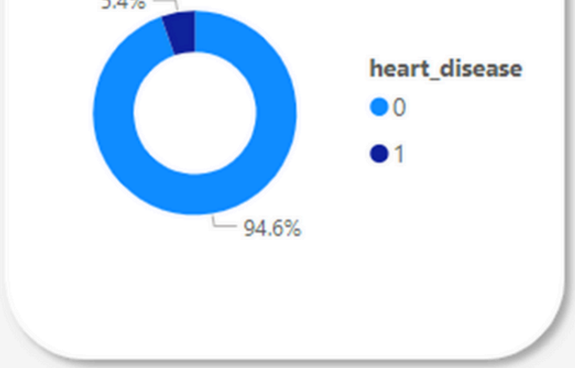
Demographics

Health Risk Factors

Lifestyle and Environment

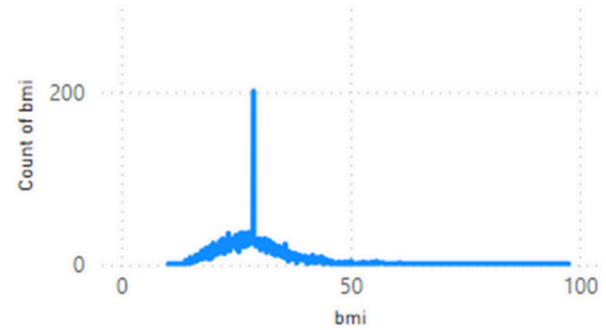
Advanced Health Metrics

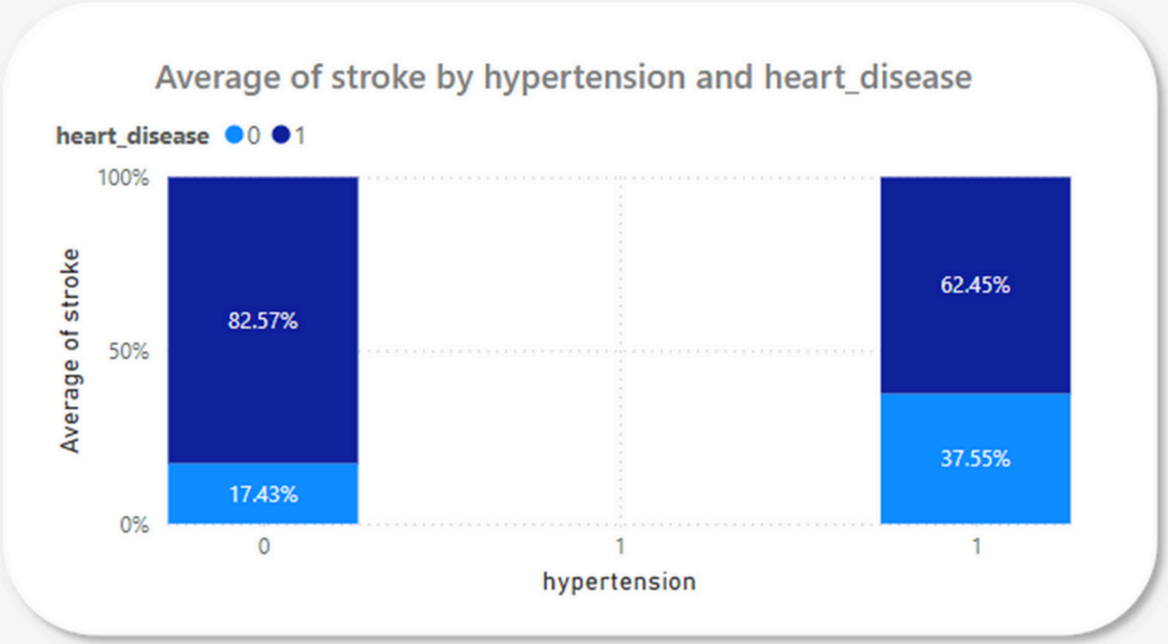


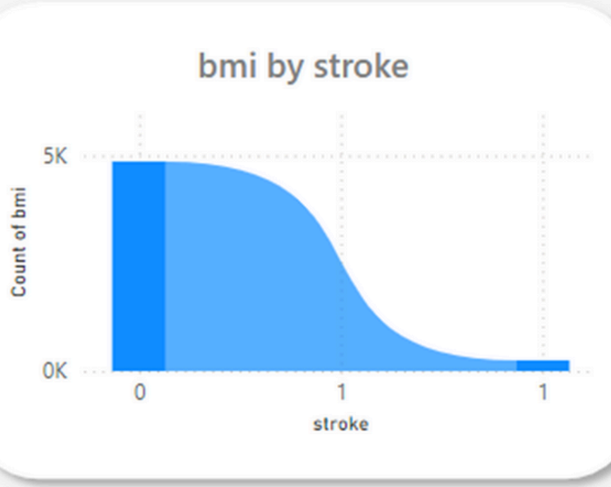


bmi Average
28.89

Max of bmi
97.60







Insights

•BMI Distribution: A bar graph displays the spread of BMI values, with an average of 28.89, indicating a slightly overweight population, and a maximum BMI of 97.60 for outliers.

•Stroke by Hypertension and Heart Disease (Bar Charts): These bar charts show that the likelihood of stroke increases significantly for those with hypertension (17.43%) and heart disease (37.55%).

•Hypertension & Heart Disease Distributions (Pie Charts): The pie charts show that 50% of the population has hypertension, while a smaller 5.4% has heart disease.

Lifestyle and Environment

Demographics

Health Risk Factors

Lifestyle and Environment

Advanced Health Metrics

Insights

•Marital Status (Bar Charts): The bar chart displays the distribution of stroke by marital status. It shows that individuals who are married (65.62%) are more likely to have had a stroke compared to those who aren’t.

•Residence Type (Pie & Bar Charts): A pie chart shows the population's distribution between urban (50.8%) and rural (49.2%) areas, with an accompanying bar chart showing a slightly higher stroke rate in urban areas.

•Work Type (Bar Chart): The bar chart shows that the majority of people work in the private sector (57.24%), followed by self-employed individuals. Stroke rates are higher among self-employed individuals and those who have never worked.

•Smoking Status (Pie & Bar Charts): A pie chart shows smoking habits: 37.03% never smoked, while a notable portion are former smokers (30.22%) or current smokers (15.44%). The bar chart shows stroke prevalence increasing among smokers.

Advanced Health Matrix

Demographics

Health Risk Factors

Lifestyle and Environment

Advanced Health Metrics

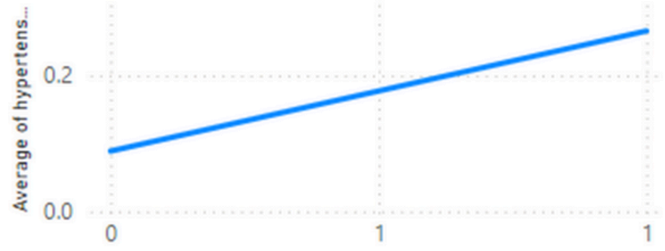
Work Types

5

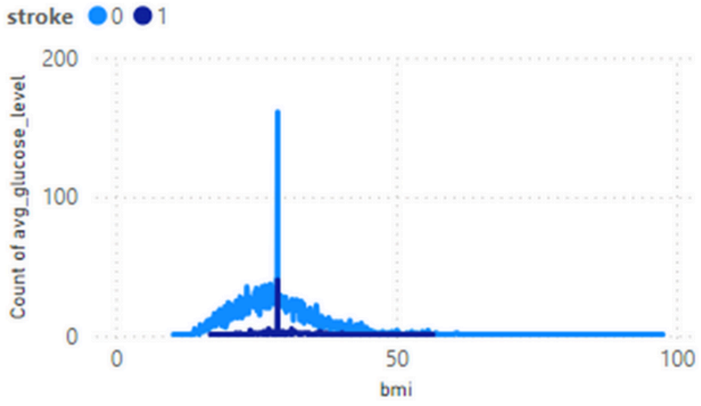
Glucose Average

271.74

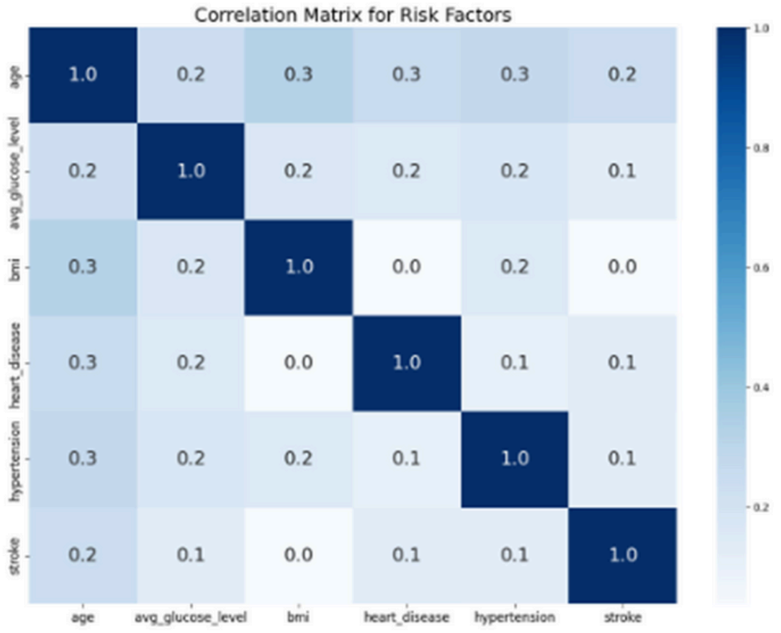
Average of hypertension by stroke



avg_glucose_level by bmi and stroke



Correlation Matrix for Risk Factors



Insights

•Correlation Matrix for Risk Factors: A heatmap-style correlation matrix highlights the relationships between various health risk factors, showing how they intersect.

•Glucose Levels by Stroke and BMI (Scatter Plot): The scatter plot reveals a high average glucose level of 271.74, with glucose levels and stroke incidents showing a potential link, especially in those with higher BMI.

•Hypertension by Stroke (Line Graph): This graph displays the average hypertension level among those who have suffered strokes, reinforcing hypertension as a strong indicator of stroke risk.

ML MODEL

Objective Of Our Model

·The objective of this project is to develop and evaluate machine learning models to accurately predict stroke occurrences in patients. Given the critical nature of stroke diagnosis, the primary focus is to minimize False Negatives (FN)—cases where a patient has a stroke, but the model incorrectly predicts no stroke. Such errors can lead to missed treatment opportunities and severe health consequences.

· The project involved exploring various machine learning algorithms and optimizing the models for high recall on the positive class (stroke cases). This ensures the model maximizes True Positives (TP), capturing as many actual stroke cases as possible.

How Our Model Could Be Used:

False Negative Avoidance: The model's focus on reducing false negatives makes it ideal for critical healthcare settings such as emergency rooms, hospitals, or clinics. It helps flag potential stroke patients, even with mild symptoms, ensuring timely attention and minimizing the risk of missed diagnoses.

Targeted Health Interventions: By accurately identifying stroke-prone individuals, healthcare providers can take preventive measures. This might include targeted health campaigns focused on smoking cessation, better diet, or managing chronic diseases.

Proactive Care Programs: Healthcare systems could use your model to allocate resources efficiently by identifying high-risk individuals and ensuring they receive proactive care. This reduces long-term healthcare costs by preventing strokes rather than treating them after they occur.

Health Apps: Patients could use a health app powered by your model to input their health data and receive personalized advice on their stroke risk, enabling them to take preventive measures.

Process to Build the Model:

01 Data Preprocessing

02 Feature Engineering

03 Data Preparation

04 Pipeline Preprocessing

05 Modeling

06 Best Model

07 Model Deployment

Data Preprocessing

1. **Checking for null values**

- **As you see here bmi column has a 201-null value so, we used KNN inputter Technique to fill these missing values**

- **there was an unknown value in the smoking_status column and to fill these values we should first replace it with NaN and then we fill NaN values using mode method**

2.Checking for duplicated rows

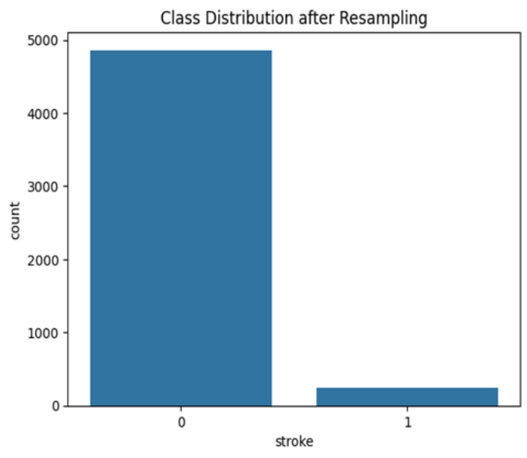
- **There weren't any duplicated rows in our dataset**

```
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
```

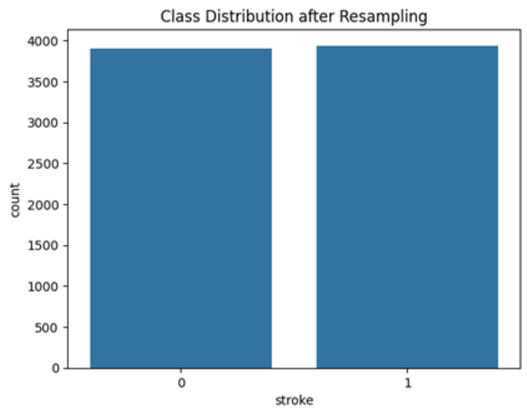
Data Preprocessing

3.Check if the data balanced or not

The chart shows a significant imbalance between the two classes in our stroke dataset. Class 0 (no stroke) has a much larger representation compared to class 1 (stroke), highlighting the challenge of predicting rare events like strokes.



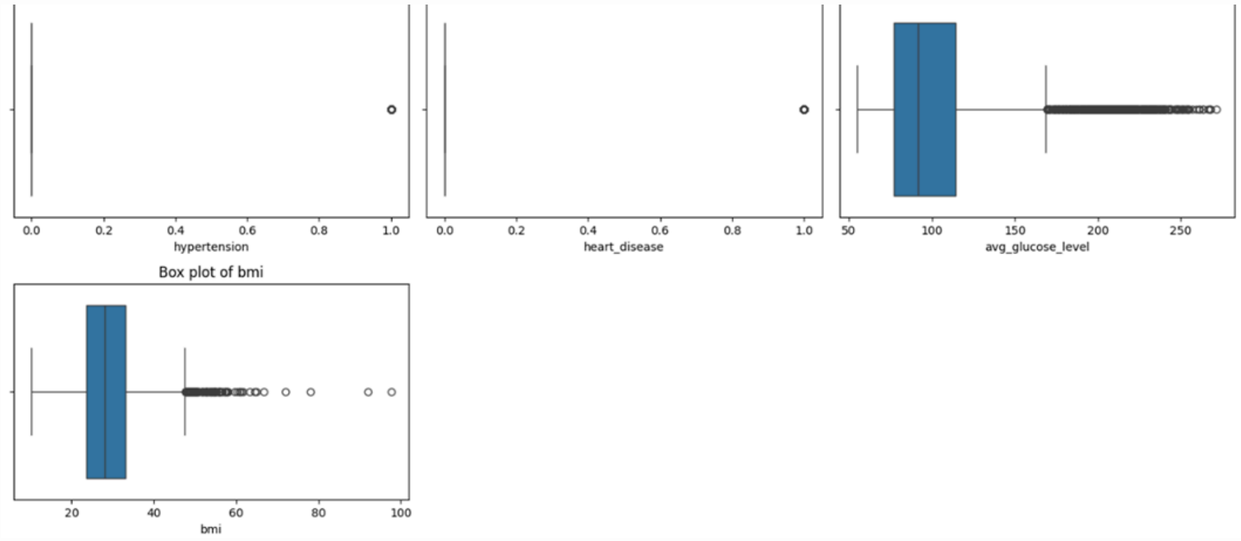
Here is the result after resampling the data using `ADASYN`



Data Preprocessing

4. Checking For outliers

"As shown in the box plots, there are noticeable outliers in the bmi and avg_glucose_level columns. Despite their presence, we decided not to remove them, as they represent a significant portion of the data. Removing these outliers could result in a loss of important information, especially since these variables are clinically relevant in stroke prediction."



Feature Engineering

In our dataset, the age column originally contained float values representing the exact age of each Individual. To enhance the model's interpretability and effectiveness, we applied feature engineering by creating an age group column, which categorized ages into meaningful bins:

Child: 0-18
Adult: 19-45
Senior: 46-60
Elderly: 61-100.
Afterward, we dropped the original age column and renamed the newly created age group column to age. Additionally, we converted its data type from category to object to better align with the categorical nature of this feature.

```
age_bins = [0, 18, 45, 60, 100]
age_labels = ['Child', 'Adult', 'Senior', 'Elderly']
df['age_group'] = pd.cut(df['age'], bins=age_bins, labels=age_labels)
```

```
# rename `age_group` column to `age` column
df.rename(columns={'age_group': 'age'}, inplace=True)

# convert it's data type from `category` to `object`
df['age'] = df['age'].astype(object)
```

Data Preparation:

- We collect all the categorical columns in a list

```
cate_columns = df.select_dtypes(include='object').columns.to_list()
```

- We collected all the numerical columns to a list, but we removed the stroke column because it’s our label

```
num_columns = df.select_dtypes(exclude='object').columns.to_list()
num_columns.remove('stroke')
```

- 1- We split our data, and we assigned our features into a variable called x and assigned our target into a variable called y

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

Preprocessing Pipeline:

- We split the data into train part and test part
- Used RobustScaler for scaling numerical features and KNN Inputer Technique for filling the missing values
- Used OneHotEncoderfor encoding categorical features.
- We made a resampling to our data using ADASYN Alhorithm
- Combined them into a single preprocessing pipeline using ColumnTransformer.

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder,  RobustScaler
from sklearn.impute import KNNImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from imblearn.combine import SMOTETomek
from imblearn.over_sampling import ADASYN
from imblearn.pipeline import Pipeline as ImbPipeline

# Split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Numerical pipeline
num_pipeline = Pipeline([
    ('numerical_inputer', KNNImputer(n_neighbors=5)),
    ('scaler', RobustScaler())
])

# Categorical pipeline
cate_pipeline = Pipeline([
    ('one_hot_encoder', OneHotEncoder(sparse_output=False, handle_unknown='ignore'))
])

# Preprocessing: combining both numerical and categorical pipelines
preprocessor = ColumnTransformer([
    ('num_pipeline', num_pipeline, num_columns),
    ('cate_pipeline', cate_pipeline, cate_columns)
]).set_output(transform='pandas')


# Full pipeline with SMOTE
model_pipeline = ImbPipeline([
    ('preprocessing', preprocessor),  # Step 1: Apply preprocessing (scaling, encoding, etc.)
    ('smote', ADASYN(random_state=42)),  # Step 2: Apply SMOTE to resample the training data

])

# Fit and resample the training data using the full pipeline
x_train_resampled, y_train_resampled = model_pipeline.fit_resample(x_train, y_train)

# Transform the test data (without applying SMOTE to test data)
x_test_preprocessed = preprocessor.transform(x_test)

# At this point, you can train your model using x_train_resampled and y_train_resampled.
```

Modeling:

Trained several models with hyperparameter tuning using GridSearchCV and evaluating performance using cross-validation, including:

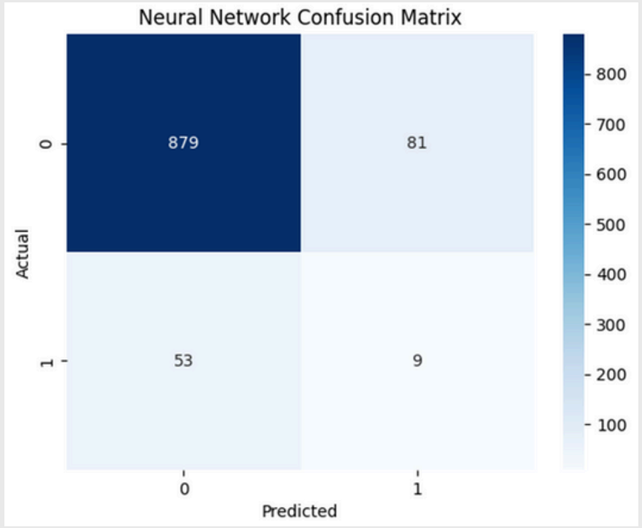
Trained several models with hyperparameter tuning using GridSearchCV and evaluating performance using cross-validation, including:

- Neural Network
- Logistic Regression
- Decision Tree
- Random Forest
- XGboost
- Support Vector Classifier

Assessed the model using accuracy, confusion matrix, and classification report to understand its precision, recall, and F1-score.

Neural Network

Confusion Matrix



Model Performance

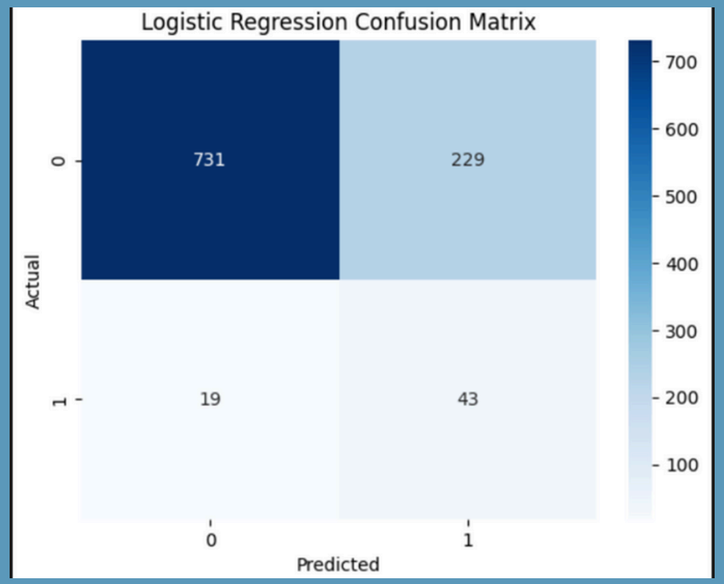
```
Classification Report: For Neural Network
              precision    recall  f1-score   support

     Class 0       0.94      0.92      0.93       960
     Class 1       0.10      0.15      0.12        62

    accuracy                           0.87      1022
   macro avg       0.52      0.53      0.52      1022
weighted avg       0.89      0.87      0.88      1022
```

Test Accuracy: 0.8689

```python
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import KFold
from tensorflow.keras.callbacks import EarlyStopping

# Prepare for K-Fold Cross-Validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# To store validation accuracies and models
val_accuracies = []
models = []

# Loop through each fold in K-Fold Cross-Validation
for train_index, val_index in kf.split(x_train_resampled):
    x_train_cv, x_val_cv = x_train_resampled.iloc[train_index], x_train_resampled.iloc[val_index]
    y_train_cv, y_val_cv = y_train_resampled.iloc[train_index], y_train_resampled.iloc[val_index]

    # Define the model
    model = keras.Sequential([
        layers.Input(shape=(x_train_cv.shape[1],)),
        layers.Dense(256, kernel_regularizer=keras.regularizers.l2(0.001)),
        layers.LeakyReLU(alpha=0.01),
        layers.BatchNormalization(),
        layers.Dropout(0.4),

        layers.Dense(128, kernel_regularizer=keras.regularizers.l2(0.001)),
        layers.LeakyReLU(alpha=0.01),
        layers.BatchNormalization(),
        layers.Dropout(0.3),

        layers.Dense(64, kernel_regularizer=keras.regularizers.l2(0.001)),
        layers.LeakyReLU(alpha=0.01),
        layers.BatchNormalization(),
        layers.Dropout(0.3),

        layers.Dense(1, activation='sigmoid')
    ])

    # Compile the model with a smaller learning rate
    model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0003),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    # Set up early stopping, monitoring val_accuracy
    early_stopping = EarlyStopping(monitor='val_accuracy', patience=15, restore_best_weights=True)

    # Train the model
    model.fit(x_train_cv, y_train_cv, epochs=200, batch_size=32,
              validation_data=(x_val_cv, y_val_cv), callbacks=[early_stopping])

    # Evaluate the model on validation set
    val_loss, val_accuracy = model.evaluate(x_val_cv, y_val_cv)
    print(f'Validation Accuracy for this fold: {val_accuracy:.4f}')

    # Store the model and its validation accuracy
    val_accuracies.append(val_accuracy)
    models.append(model)

# Calculate the average validation accuracy across all folds
average_accuracy = np.mean(val_accuracies)
print(f'Average Validation Accuracy: {average_accuracy:.4f}')

# Find the model closest to the average accuracy
closest_index = np.argmin(np.abs(np.array(val_accuracies) - average_accuracy))
best_model = models[closest_index]
print(f'Model closest to the average accuracy is from fold {closest_index + 1}.')

# Train the selected model on the entire resampled training data
best_model.fit(x_train_resampled, y_train_resampled, epochs=300, batch_size=32, callbacks=[early_stopping])

# Evaluate the best model on the test data
test_loss, test_accuracy = best_model.evaluate(x_test_preprocessed, y_test)
print(f'Test accuracy: {test_accuracy:.4f}')

# Make predictions on test data
predictions = best_model.predict(x_test_preprocessed)
predicted_classes = (predictions > 0.5).astype("int32")

```

1-Logistic Regression

Confusion Matrix



Model Performance

```
print("\nClassification Report:")
print(classification_report(y_test, y_pred_threshold))
```

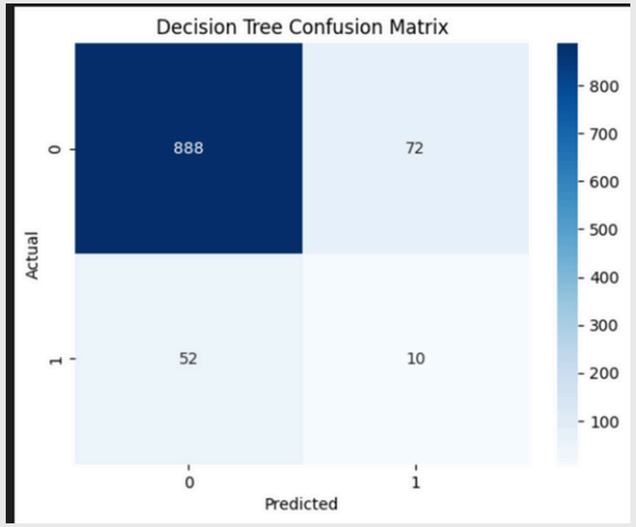
```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.76      0.85       960
           1       0.16      0.69      0.26        62

    accuracy                           0.76      1022
   macro avg       0.57      0.73      0.56      1022
weighted avg       0.93      0.76      0.82      1022
```

Accuarcy=0.7573385

```
# 1. Import Required Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    roc_auc_score,
    precision_recall_curve
)
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import seaborn as sns




# 7. Initialize Logistic Regression with Class Weights
# Class weights help the model pay more attention to the minority class
logreg = LogisticRegression(class_weight='balanced', solver='liblinear', max_iter=300)

# 8. Define Hyperparameter Grid
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],             # Inverse of regularization strength
    'penalty': ['l1', 'l2'],                  # Regularization penalties
    'solver': ['liblinear', 'saga'],          # Solvers compatible with penalties
    'max_iter': [300, 400, 500]               # Maximum iterations for convergence
}

# 9. Initialize GridSearchCV
# Focus on optimizing recall to reduce False Negatives
grid_search = GridSearchCV(
    estimator=logreg,
    param_grid=param_grid,
    scoring='recall',           # Optimize for recall
    cv=5,                       # 5-fold cross-validation
    n_jobs=-1,                  # Use all available cores
    verbose=2
)

# 10. Fit GridSearchCV on Resampled Training Data
grid_search.fit(x_train_resampled, y_train_resampled)

# 11. Retrieve Best Parameters and Best Estimator
best_params = grid_search.best_params_
LRG_Model = grid_search.best_estimator_

print("Best Hyperparameters (Optimized for Recall):")
print(best_params)

# 12. Make Predictions on Test Set
y_pred = LRG_Model.predict(x_test_preprocessed)
y_prob = LRG_Model.predict_proba(x_test_preprocessed)[:, 1]  # Probability estimates for the positive class

# 13. Adjust Classification Threshold to Further Reduce False Negatives
# Lowering the threshold increases recall but may affect precision
threshold = 0.6  # Example threshold; you can adjust based on the trade-off
y_pred_threshold = (y_prob >= threshold).astype(int)


accuracy_LRG = accuracy_score(y_test, y_pred_threshold)
print(f"Logistic REgression Accuracy: {accuracy_LRG}")
```

2-Decision Tree:

Confusion Matrix



Model Performance

Classification Report:

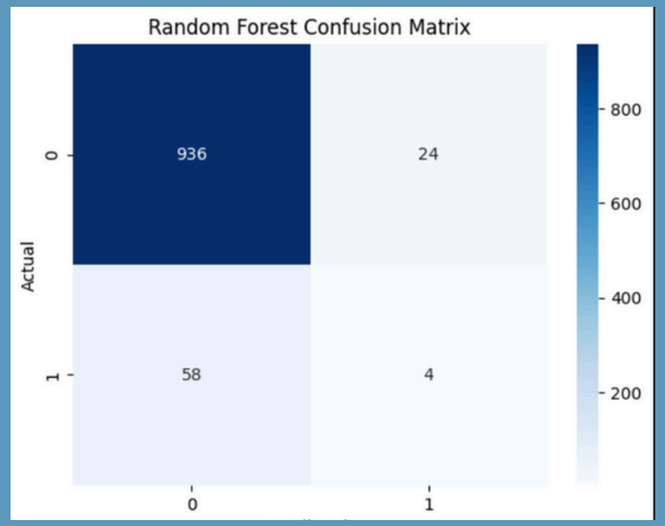
	precision	recall	f1-score	support
0	0.94	0.93	0.93	960
1	0.12	0.16	0.14	62
accuracy			0.88	1022
macro avg	0.53	0.54	0.54	1022
weighted avg	0.89	0.88	0.89	1022

Accuarcy=0.87866

```python
# 1. Import Required Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    roc_auc_score,
    precision_recall_curve
)
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import seaborn as sns




# 7. Initialize Logistic Regression with Class Weights
# Class weights help the model pay more attention to the minority class
logreg = LogisticRegression(class_weight='balanced', solver='liblinear', max_iter=300)

# 8. Define Hyperparameter Grid
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],             # Inverse of regularization strength
    'penalty': ['l1', 'l2'],                  # Regularization penalties
    'solver': ['liblinear', 'saga'],          # Solvers compatible with penalties
    'max_iter': [300, 400, 500]               # Maximum iterations for convergence
}

# 9. Initialize GridSearchCV
# Focus on optimizing recall to reduce False Negatives
grid_search = GridSearchCV(
    estimator=logreg,
    param_grid=param_grid,
    scoring='recall',            # Optimize for recall
    cv=5,                        # 5-fold cross-validation
    n_jobs=-1,                   # Use all available cores
    verbose=2
)

# 10. Fit GridSearchCV on Resampled Training Data
grid_search.fit(x_train_resampled, y_train_resampled)

# 11. Retrieve Best Parameters and Best Estimator
best_params = grid_search.best_params_
LRG_Model = grid_search.best_estimator_

print("Best Hyperparameters (Optimized for Recall):")
print(best_params)

# 12. Make Predictions on Test Set
y_pred = LRG_Model.predict(x_test_preprocessed)
y_prob = LRG_Model.predict_proba(x_test_preprocessed)[:, 1]  # Probability estimates for the positive class

# 13. Adjust Classification Threshold to Further Reduce False Negatives
# Lowering the threshold increases recall but may affect precision
threshold = 0.6  # Example threshold; you can adjust based on the trade-off
y_pred_threshold = (y_prob >= threshold).astype(int)


accuracy_LRG = accuracy_score(y_test, y_pred_threshold)
print(f"Logistic REgression Accuracy: {accuracy_LRG}")
```

3-Random forest

Confusion matrix:



Model Performance

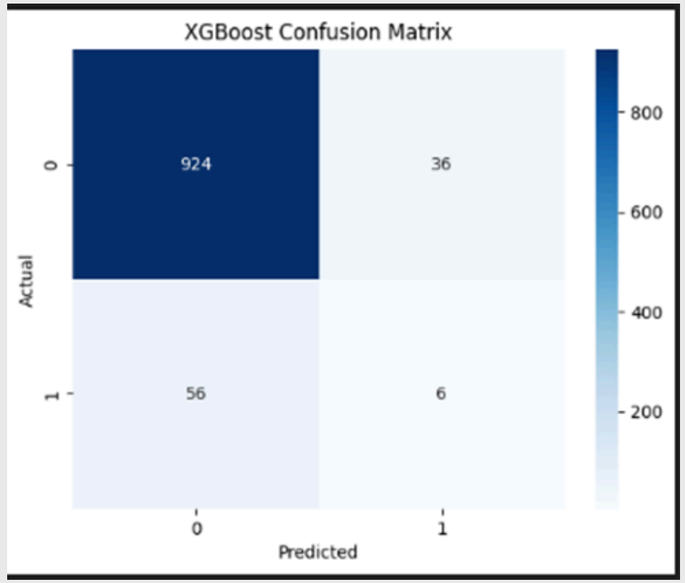
	precision	recall	f1-score	support
0	0.94	0.97	0.96	960
1	0.14	0.06	0.09	62
accuracy			0.92	1022
macro avg	0.54	0.52	0.52	1022
weighted avg	0.89	0.92	0.91	1022

Model Accuray = .9197651663

```python
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV, KFold, cross_val_score
from sklearn.metrics import (
    accuracy_score, classification_report, confusion_matrix, roc_auc_score
)

# 1. Define Hyperparameter Grid (Reduced Search Space)
param_grid = {
    'n_estimators': [100, 150, 200],  # Fewer values for faster tuning
    'max_depth': [10, 20, None],  # Include None for no depth limit
    'min_samples_split': [2, 5],  # Reasonable range
    'min_samples_leaf': [1, 2],   # Only minimal range
    'max_features': ['sqrt', 'log2']  # Avoid None for efficiency
}

# 2. Initialize Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42)

# 3. Set up RandomizedSearchCV for Faster Search
cv = KFold(n_splits=3, shuffle=True, random_state=42)
random_search_rf = RandomizedSearchCV(
    estimator=rf_model,
    param_distributions=param_grid,
    n_iter=10,  # Limit to 10 parameter combinations
    scoring='recall',  # Optimize for recall
    cv=cv,
    n_jobs=-1,  # Use all available CPU cores
    verbose=1,
    random_state=42
)

# 4. Fit RandomizedSearchCV on Resampled Data
random_search_rf.fit(x_train_resampled, y_train_resampled)

# 5. Retrieve Best Model and Parameters
best_params_rf = random_search_rf.best_params_
best_rf_model = random_search_rf.best_estimator_

print("Best Hyperparameters (Optimized for Recall):")
print(best_params_rf)

# 6. Train Accuracy on Resampled Data
train_accuracy = accuracy_score(y_train_resampled, best_rf_model.predict(x_train_resampled))
print(f"Train Accuracy: {train_accuracy}")

# 7. Cross-Validation Performance
scores_rf = cross_val_score(best_rf_model, x_train_resampled, y_train_resampled, cv=cv, scoring='recall')
print(f"Random Forest Cross-validation scores (Recall): {scores_rf}")
print(f"Mean Recall: {np.mean(scores_rf)}")
print(f"Standard Deviation: {np.std(scores_rf)}")

# 8. Make Predictions on Test Data Using Threshold
# Get the predicted probabilities for the positive class (class 1)
y_prob_rf = best_rf_model.predict_proba(x_test_preprocessed)[:, 1]

# Apply the threshold of 0.6 to convert probabilities into binary predictions
threshold = 0.6
y_pred_rf_threshold = np.where(y_prob_rf >= threshold, 1, 0)
```

4-Xg Boost:

Confusion Matrix



Model Performance

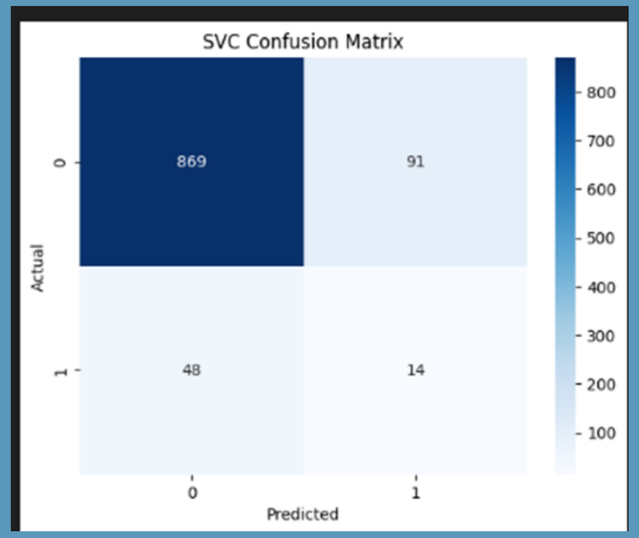
Classification Report:	precision	recall	f1-score	support
0	0.94	0.96	0.95	960
1	0.14	0.10	0.12	62
accuracy			0.91	1022
macro avg	0.54	0.53	0.53	1022
weighted avg	0.89	0.91	0.90	1022

Accuarcy=0.9099

```python
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV, KFold
from sklearn.metrics import (
    accuracy_score, classification_report, confusion_matrix, precision_recall_curve
)
import numpy as np

# 1. Define Hyperparameter Distribution
param_distributions = {
    'n_estimators': [100, 200, 300, 400],      # Number of trees
    'max_depth': [3, 5, 10],                   # Control tree depth
    'learning_rate': [0.01, 0.1, 0.2],         # Step size shrinkage
    'subsample': [0.5, 0.6, 0.7, 0.8],         # Fraction of samples used for fitting
    'colsample_bytree': [0.5, 0.6, 0.7],       # Fraction of features used for fitting
    'gamma': [0, 0.1],                         # Minimum loss reduction for a split
    'reg_alpha': [0, 0.1]                      # L1 regularization
}

# 2. Initialize XGBoost Classifier
xg_model = XGBClassifier(random_state=42)

# 3. Initialize RandomizedSearchCV
cv = KFold(n_splits=5, shuffle=True, random_state=42)
randomized_search_xg = RandomizedSearchCV(
    estimator=xg_model,
    param_distributions=param_distributions,
    n_iter=50,  # Number of random combinations to try
    scoring='recall',  # Optimize for recall to reduce False Negatives
    cv=cv,
    n_jobs=-1,
    verbose=2,
    random_state=42
)

# 4. Fit RandomizedSearchCV on Resampled Data
randomized_search_xg.fit(x_train_resampled, y_train_resampled)

# 5. Retrieve the Best Model and Parameters
best_params_xg = randomized_search_xg.best_params_
best_xg_model = randomized_search_xg.best_estimator_

print("Best Hyperparameters (Optimized for Recall):")
print(best_params_xg)

# 6. Make Predictions on Test Data
y_prob_xg = best_xg_model.predict_proba(x_test_preprocessed)[:, 1]  # Probability estimates

# 7. Adjust Threshold to Further Reduce False Negatives
threshold = 0.6  # Example threshold, can be tuned
y_pred_threshold_xg = (y_prob_xg >= threshold).astype(int)

accuracy_xg = accuracy_score(y_test, y_pred_threshold_xg)
print(f"XGboost Accuracy: {accuracy_xg}")
```

5-svm(Support Vector Machine):

Confusion matrix:



Model Performance

Classification Report:	precision	recall	f1-score	support
0	0.95	0.91	0.93	960
1	0.13	0.23	0.17	62
accuracy			0.86	1022
macro avg	0.54	0.57	0.55	1022
weighted avg	0.90	0.86	0.88	1022

Model Accuray = .9307

```python
import numpy as np
from sklearn.svm import SVC, LinearSVC
from sklearn.model_selection import RandomizedSearchCV, KFold, cross_val_score
from sklearn.metrics import accuracy_score
from scipy.stats import uniform

# 1. Define Narrowed Hyperparameter Distributions for Randomized Search
param_dist = {
    'C': uniform(0.1, 10),  # Narrowed range for C
    'kernel': ['linear', 'rbf'],  # Considering linear and rbf kernels
    'gamma': ['scale', 'auto'],  # Kernel coefficient options
    'coef0': uniform(0.0, 0.1)  # Narrowed range for coef0
}

# 2. Initialize SVC Classifier (or LinearSVC for linear kernel)
svc_model = SVC(random_state=42, probability=True)  # Probability estimates enabled for threshold tuning

# 3. Initialize RandomizedSearchCV with Reduced Parameters
cv = KFold(n_splits=3, shuffle=True, random_state=42)  # Reduced K-Folds to 3 for faster evaluation
random_search_svc = RandomizedSearchCV(
    estimator=svc_model,
    param_distributions=param_dist,
    scoring='recall',  # Optimizing for recall
    cv=cv,
    n_iter=10,  # Reduced number of iterations for faster search
    n_jobs=-1,  # Parallelize across all available CPUs
    verbose=2,
    random_state=42
)

# 4. Fit RandomizedSearchCV on Resampled Data
random_search_svc.fit(x_train_resampled, y_train_resampled)

# 5. Retrieve the Best Model and Parameters
best_params_svc = random_search_svc.best_params_
best_svc_model = random_search_svc.best_estimator_

print("Best Hyperparameters (Optimized for Recall):")
print(best_params_svc)

# 6. Evaluate Performance on Training Data
train_accuracy = accuracy_score(y_train_resampled, best_svc_model.predict(x_train_resampled))
print(f"Train Accuracy: {train_accuracy}")

# 7. Cross-Validation Performance
scores = cross_val_score(best_svc_model, x_train_resampled, y_train_resampled, cv=cv, scoring='recall')
print(f"SVC Cross-validation scores (Recall): {scores}")
print(f"Mean Recall: {np.mean(scores)}")
print(f"Standard Deviation: {np.std(scores)}")

# 8. Make Predictions on Test Data
y_prob_svc = best_svc_model.predict_proba(x_test_preprocessed)[:, 1]  # Probability estimates

# 9. Adjust Threshold to Further Reduce False Negatives
threshold = 0.6  # Example threshold, can be tuned
y_pred_threshold_svc = (y_prob_svc >= threshold).astype(int)
```

Best Model:

Logistic Regression: The Best Model for Stroke Prediction

After evaluating multiple models for stroke prediction, Logistic Regression emerged as the best-performing model for several reasons:

01

Balanced Class Handling: Logistic Regression, with class weights, effectively addresses class imbalance by giving more importance to the minority class (stroke cases), improving recall.

02

Recall Optimization: The project prioritizes recall to minimize false negatives in stroke prediction. Logistic Regression, with hyperparameter tuning, outperformed other models like Random Forest and XGBoost in identifying stroke cases.

03

Interpretability: Logistic Regression's interpretable coefficients provide insights into how various factors influence stroke likelihood, which is valuable in medical contexts.

04

Model Performance: Despite not having the highest accuracy, Logistic Regression achieved a good balance between precision and recall, particularly improving recall for stroke cases.

05

Simplicity and Efficiency: Logistic Regression is computationally efficient, works well with smaller datasets, and generalizes effectively, avoiding overfitting based on cross-validation results.

Model Deployment

We created an interactive web application using Steramlit where users can add:

- Input Customer
- View the model Prediction on whether the patient would have stroke or not

DOCKER

Why docker ?

It is an open -source platform used for Containerisation, allowing you to package an application and its dependencies into a standardized unit which is a container.

Docker Engine:

The core component of Docker, responsible for running and managing containers on a host system.

Docker Client:

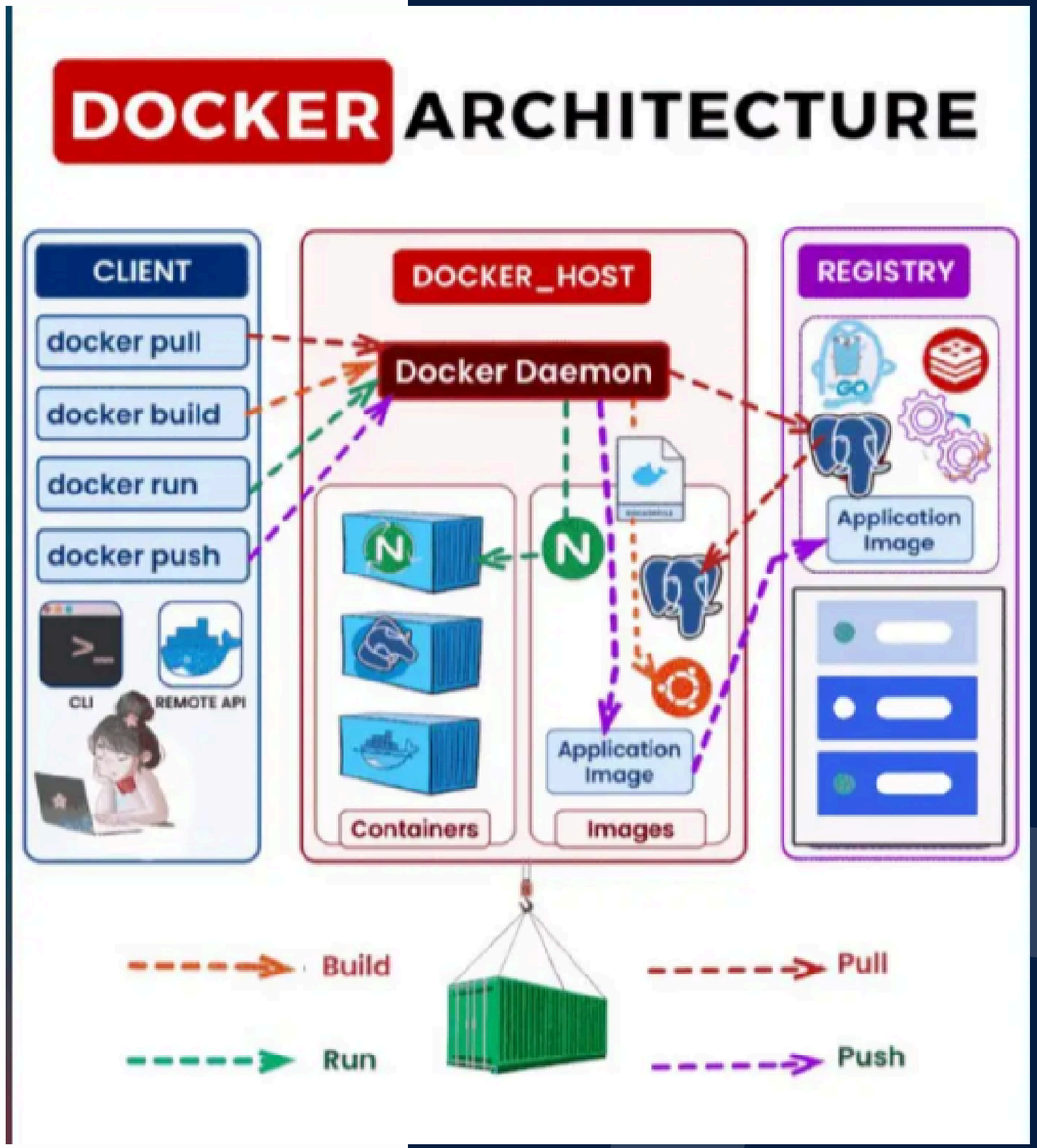
The command-line interface (CLI) that users interact with to send commands to the Docker Daemon, such as building, running, and stopping containers.

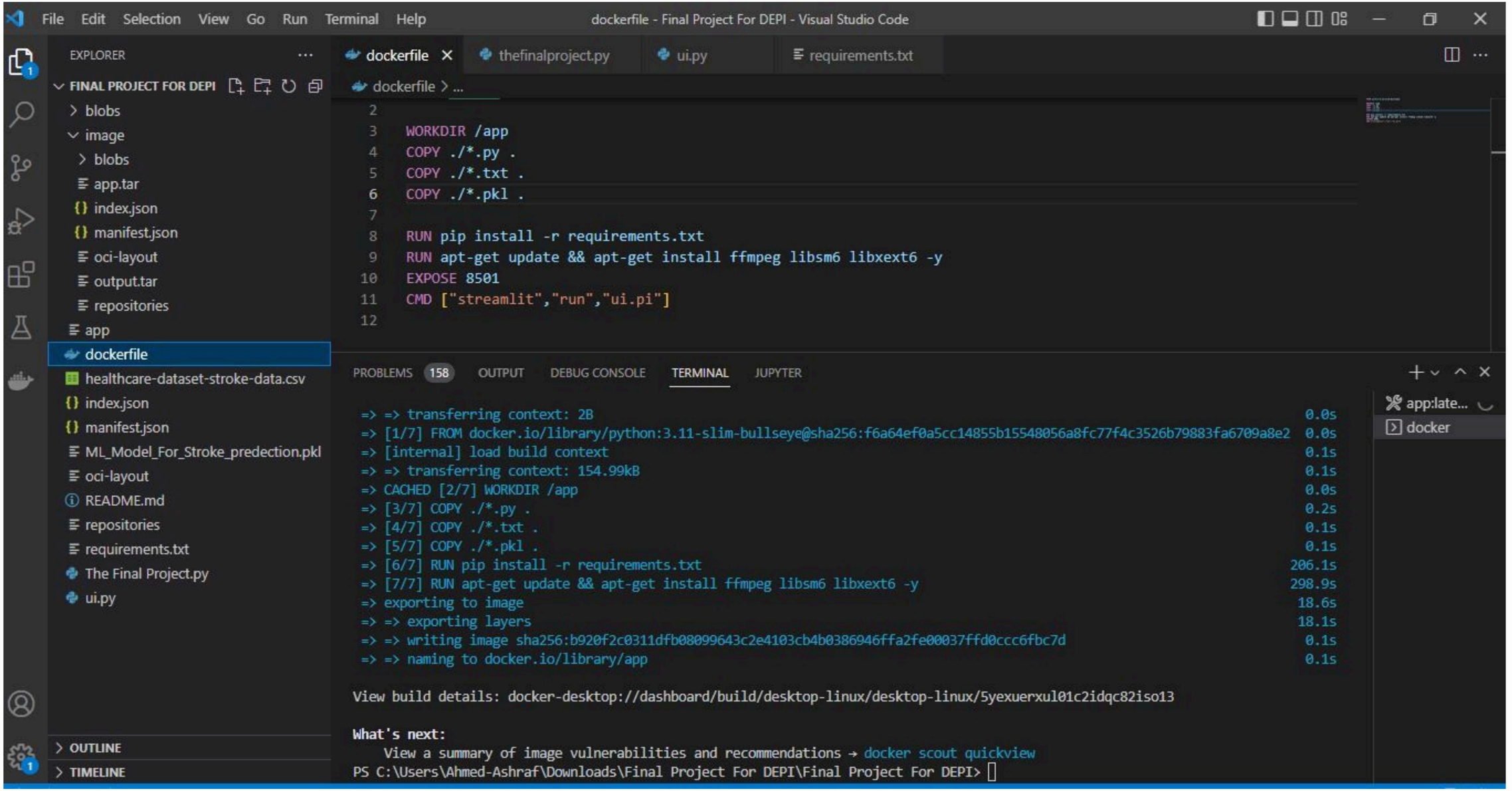
Docker Daemon:

The background service that runs on the host, managing Docker containers, images, networks, and volumes, and listening for commands from the Docker Client.

Docker Registry:

A centralized storage location where Docker images are stored, from which users can push and pull images, with Docker Hub being the default public registry.




File Edit Selection View Go Run Terminal Help
dockerfile - Final Project For DEPI - Visual Studio Code
EXPLORER
FINAL PROJECT FOR DEPI
blobs
image
blobs
app.tar
index.json
manifest.json
oci-layout
output.tar
repositories
app
dockerfile
healthcare-dataset-stroke-data.csv
index.json
manifest.json
ML_Model_For_Stroke_predection.pkl
oci-layout
README.md
repositories
requirements.txt
The Final Project.py
ui.py
OUTLINE
TIMELINE
dockerfile
thefinalproject.py
ui.py
requirements.txt
dockerfile > ...
2
3 WORKDIR /app
4 COPY ./*.py .
5 COPY ./*.txt .
6 COPY ./*.pkl .
7
8 RUN pip install -r requirements.txt
9 RUN apt-get update && apt-get install ffmpeg libsm6 libxext6 -y
10 EXPOSE 8501
11 CMD ["streamlit","run","ui.pi"]
12
PROBLEMS 158
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER
=> => transferring context: 2B 0.0s
=> [1/7] FROM docker.io/library/python:3.11-slim-bullseye@sha256:f6a64ef0a5cc14855b15548056a8fc77f4c3526b79883fa6709a8e2 0.0s
=> [internal] load build context 0.1s
=> => transferring context: 154.99kB 0.1s
=> CACHED [2/7] WORKDIR /app 0.0s
=> [3/7] COPY ./*.py . 0.2s
=> [4/7] COPY ./*.txt . 0.1s
=> [5/7] COPY ./*.pkl . 0.1s
=> [6/7] RUN pip install -r requirements.txt 206.1s
=> [7/7] RUN apt-get update && apt-get install ffmpeg libsm6 libxext6 -y 298.9s
=> exporting to image 18.6s
=> => exporting layers 18.1s
=> => writing image sha256:b920f2c0311dfb08099643c2e4103cb4b0386946ffa2fe00037ffd0ccc6fbc7d 0.1s
=> => naming to docker.io/library/app 0.1s
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/5yexuerxul01c2idqc82iso13
What's next:
View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\Ahmed-Ashraf\Downloads\Final Project For DEPI\Final Project For DEPI>
app:late...
docker

Advantages

01 Portability

02 Consistency

03 Isolation

04 Scalability

Our Team

Mahmoud Ahmed

Weam Taie

Rehab Yehia

Nada Mohamed

Kamal Khaled

Mohamed Fouda

THANK YOU!

View The Project Web Application From Here:

Streamlit

Github link: