Data Science 2025-1

Team Project Result Report

**Stroke Risk Prediction Based on**

**Patient Health Records**

Team D

202235038 문희진

202235070 안수빈

202235130 조혜린

202033624 한민섭

2025-6-2

**1.** **Executive Summary**

Stroke is a leading cause of death and disability worldwide. Therefore, early diagnosis of individuals at high risk can prevent severe outcomes. Our objective is to develop a classification model that predicts the occurrence of stroke based on information from patients who have experienced a stroke and those who have not.

Based on our experimental results, the best-performing model identified through 8-Fold Cross-Validation was **RandomForest\_ver1** (with n\_estimators=200 and max\_depth=20), achieving an 89% ACC and 73% ROC AUC. This model demonstrates the potential to support clinical decision-making by assisting physicians in the early identification of high-risk stroke patients through inputting basic patient information. It suggests a practical direction for implementing AI-based medical support systems, potentially contributing to more efficient diagnosis, as well as reductions in both time and cost.

**2.** **Introduction**

Due to the significantly lower number of abnormal cases, which are the primary focus of stroke analysis, compared to normal cases in our dataset, we utilized SMOTE (Synthetic Minority Over-sampling Technique) to generate synthetic data corresponding to stroke cases. This synthetic data was created based on useful data obtained through data preprocessing.

To improve the accuracy of the analysis, data pre-processing steps were applied to key variables. Missing values in the **bmi** feature were imputed using the **median**, as it provides the accuracy of the analysis . In the case of the **age** variable, extreme values were processed as outliers and subsequently removed to prevent distortion of the model's learning process. Also in **gender** features, only one outlier “Other” gender data but not a stroke(normal patient) existed so we decided to ignore this value.

For categorical features, **One-hot Encoding** was applied to convert nominal values into a numerical format suitable for machine learning algorithms. Additionally, **StandardScaler** was used to normalize all numerical features, ensuring they are on a comparable scale and enhancing the stability of the model training process.

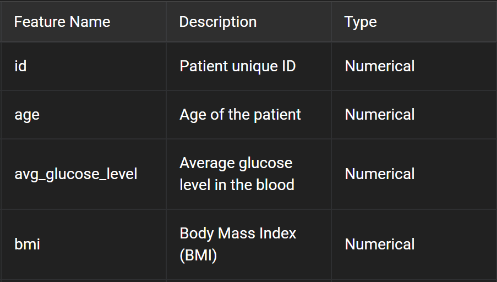
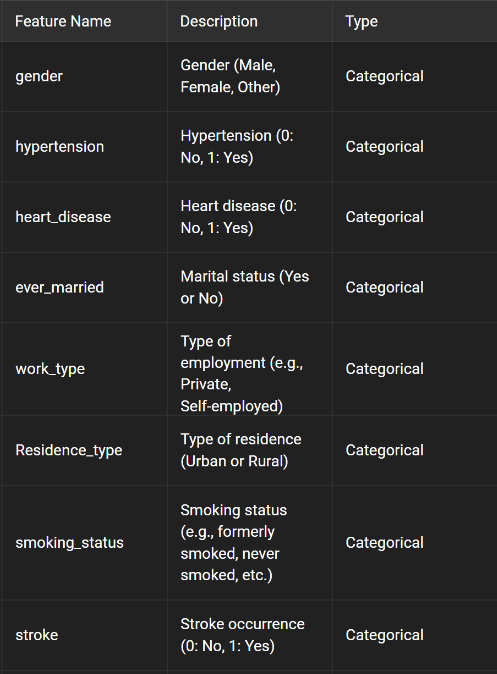
To analyze which feature is influential to the stroke, we use RandomForestClassifier to see feature importance. With K-Fold Cross Validation, we implemented finding optimal K fold as loop, and when K=8, the highest ACC and ROC AUC came out.

Results of Clustering and K-Means are visualized by PCA(Primary Component Analysis). After a performance comparison of five(knn, 2 kinds of logistic regression models, 2 kinds of random forest models) classification models, we decided to use Random Forest with 200 numbers of estimators and depth 20.

**3.** **Data Description**

The data source from Kaggle(stroke prediction dataset), based on 5,110 patient entries.

In attributes, “gender”, “hypertension”, “heart\_disease”, “ever\_married”, “work\_type”, “Residence\_type”, and “smoking status”. For numerical data, “age”, “avg\_glucose\_level”, and “bmi” have been used. And **the target is stroke**(0: no, 1: yes).



**4.** **Results**

We structured our project around a fully modular and reusable codebase that adheres to full data science pipelines. Our modular code structure allows seamless integration and evaluation of various models by dynamically importing them from external scripts. Below is a summary of our methodology.

**Data Inspection(data inspection function):**

Checked for missing values in all columns and confirmed the distribution of the target variable (stroke), identifying a strong class imbalance (positive class < 5%).

Visualized potential outliers using scatter plots:

age vs bmi

age vs avg\_glucose\_level

**Data Preprocessing (preprocess\_data function):**

Handled missing values in bmi using median imputation.

Filtering:

Removed rows with gender = Other

Excluded patients under age 20

Removed outliers with bmi >= 80

Categorical Encoding: Applied One-Hot Encoding with drop\_first=True to reduce multicollinearity.

Numerical Scaling: Standardized continuous features using StandardScaler.

**Synthetic Oversampling:**

Employed SMOTE to balance the dataset, generating synthetic minority class (stroke) samples during each fold in cross-validation.

**Model Setup (load\_models\_from\_files function):**

Loaded five models via dynamic import:

knn.py

* K-Nearest Neighbors classifier with fixed n\_neighbors=13, and varies the distance metric (p) and weighting (uniform, distance)

LogisticRegression\_ver1.py

* Logistic Regression (C=1, penalty='l2'), A stable, full-training model that uses all features.

LogisticRegression\_ver2.py

* Logistic Regression (C=0.1, penalty='l1') for feature selection and sparsity. Similar flow to ver1, but better suited for reducing complexity by excluding unimportant features. Efficient and interpretable model that achieves similar performance even after removing some features.

RandomForest\_ver1.py

* Random Forest model with 200 estimators and depth limited to 20 (max\_depth=20). Utilizes all features and builds a complex model. Risk of overfitting and difficult to interpret, but capable of capturing intricate patterns.

RandomForest\_ver2.py

* Random Forest without limiting depth (max\_depth=None), allowing deep trees to potentially capture complex patterns. May experience slight performance drop. Improves interpretability and is expected to generalize better.

Adjusted model hyperparameters within code for each version.

**Optimal K Determination (kfoldcross.py)**

We evaluated multiple k values for Stratified K-Fold cross-validation using a separate module. The optimal k was selected based on maximum average ROC AUC.

stable model performance, balanced class distribution in each fold, reproducible validation results

**Model Evaluation (cross\_validate\_model function):**

Performed Stratified K-Fold Cross-Validation (K=8) to maintain label balance across folds.

Metrics collected:

* Accuracy
* ROC AUC

Models were compared based on a composite score: 0.6 \* ROC AUC + 0.4 \* Accuracy

Final Model Selection:

RandomForest (ver2) with n\_estimators=200, max\_depth=20 selected as the most practical and robust classifier for stroke prediction.

**Clustering Analysis (K-Means + PCA)**

To explore latent structures:

Performed K-Means clustering on the resampled & scaled dataset.

Chose optimal number of clusters (K) using Silhouette Score.

Applied PCA to reduce to 2D space for visualization.

Analyzed stroke ratios per cluster to identify high-risk subgroups.

Visualization confirmed distinguishable clusters with different stroke probabilities, supporting the idea that similar profiles share risk patterns.

**End-to-End Pipeline Execution**

All components were unified under the if \_\_name\_\_ == '\_\_main\_\_' block to ensure:

Full pipeline integration (from preprocessing → model selection → prediction)

One-click execution and reproducibility

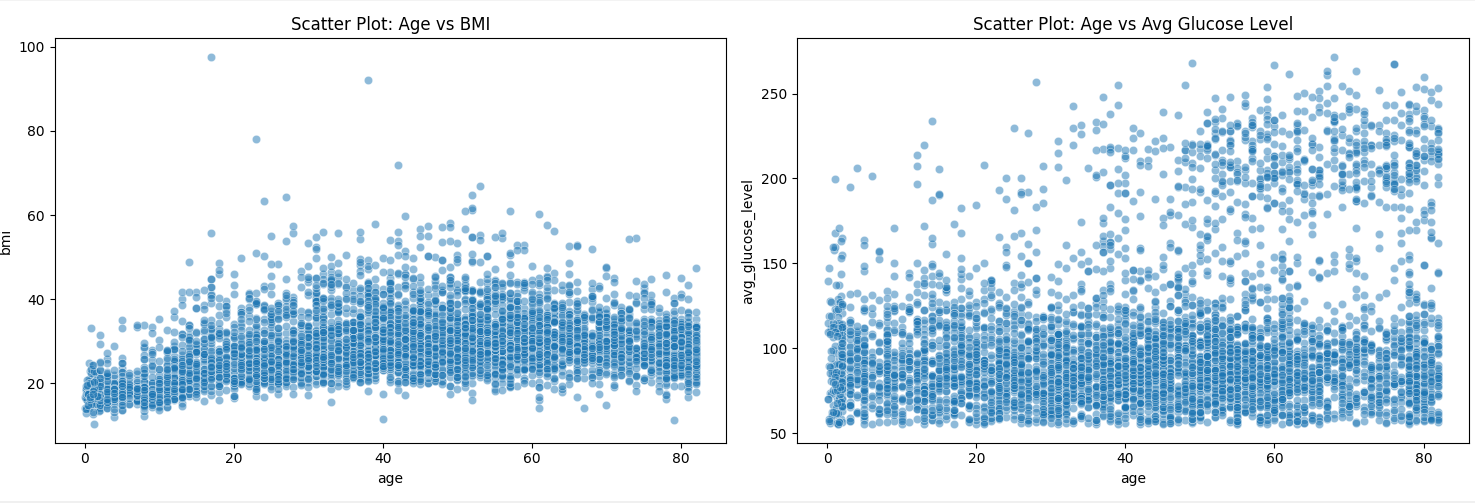
Real-time adaptability with new input

**Prediction for New Patients:**

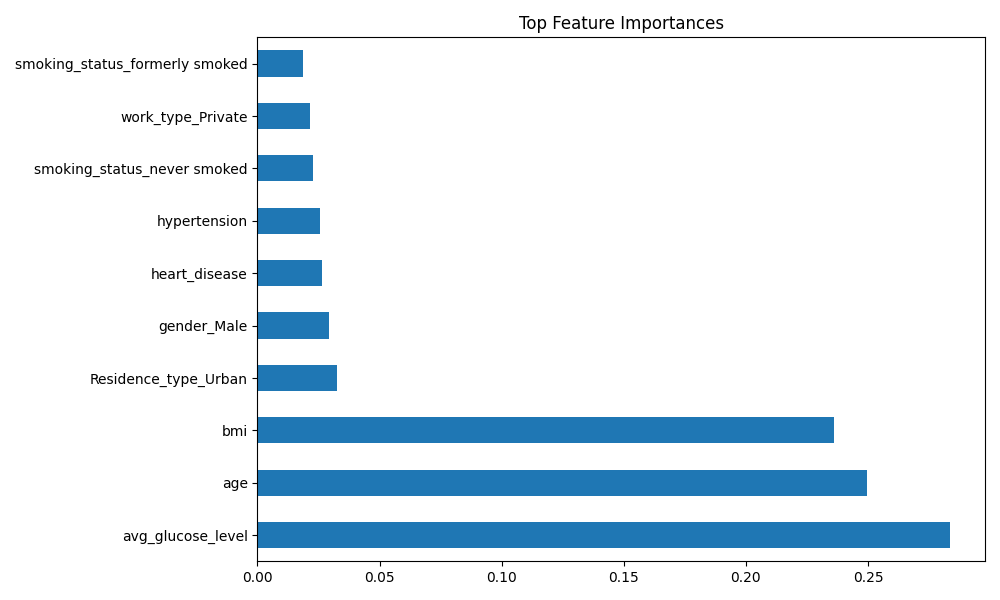
Implemented input\_patient\_data() and preprocess\_new\_data() functions. Takes basic user information, processes it in real-time, and returns stroke probability using the final trained model.

**Visualization Outputs:**

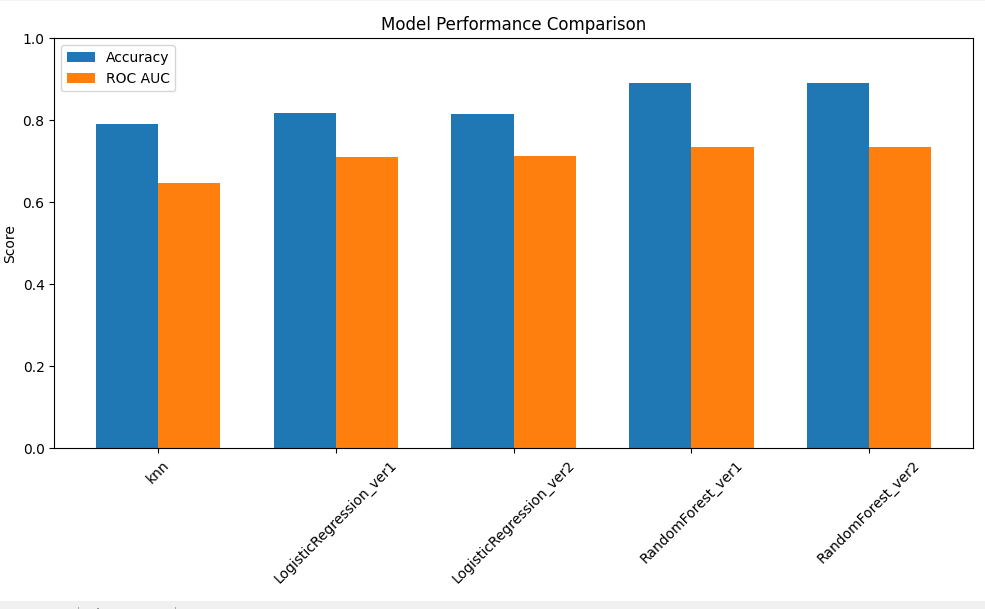
Scatter Plots (Age vs BMI, Age vs Glucose Level)



Feature Importance (via Random Forest) PCA-transformed Clustering View



Model performance comparison



**Limitations & Future Work**

Missing features: No blood pressure, cholesterol, or genetic information present in current dataset.

SMOTE caveat: Synthetic data may not always reflect clinical variability.

Model explainability: Future versions may include SHAP or LIME to visualize feature influence.

Threshold tuning: Probability cutoff (0.5) can be adjusted to optimize recall in healthcare-sensitive use cases.

**5. Discussion**

1. **Label encoding** is a method that converts unique string values into numeric values. Initially, we considered label encoding to numerically transform the data. However, due to the large number of categorical attributes and the need to assign values from 0 to N for each attribute, the process was expected to become complicated. Moreover, since our classification task does not rely on any inherent order, label encoding could mistakenly introduce ordinal characteristics by assigning numerical values, potentially implying relationships or weights that do not exist. This raised concerns, especially because such misinterpretations could critically affect **K-Means Clustering**, which we employ in our model. Considering these issues and the potential degradation in model performance, we decided to switch to **One-hot Encoding** instead.

One-Hot Encoding avoids this by treating each category as a separate binary feature, making it more suitable and reliable for clustering.

1. The BMI range includes categories such as underweight and normal weight, which are globally accepted standards. When we researched the standard, we found that most countries apply BMI classification only to individuals aged 20 or older. Therefore, we excluded patients under the age of 20 from our dataset.

Out of 996 patients under 20, only 2 had experienced a stroke, making the proportion extremely small. As a result, we decided to ignore this group and focus only on patients aged over 20. Also, we considered that the cause of stroke in patients under the age of 20 might be due to accidental events or extremely rare cases that are not typically observed in adults.

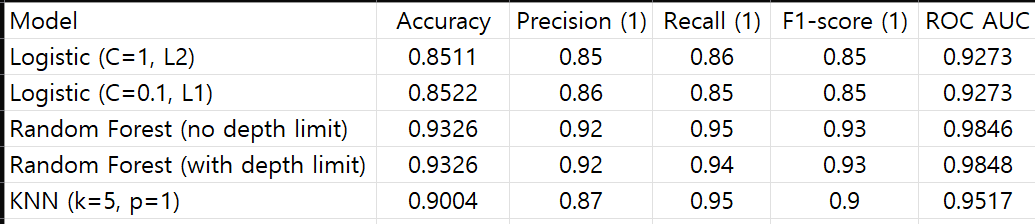
1. Based on the results of testing and comparing various combinations, we selected five combinations of learning model training and testing. **KNN** is a distance-based algorithm that can effectively classify data based on meaningful distances in numerical features. Its performance becomes more stable when class imbalance is addressed through **SMOTE**.

**Logistic Regression** allows us to interpret the influence of each feature on stroke occurrence through its coefficients. Applying L1 or L2 regularization helps prevent overfitting and enables variable selection. The model is particularly useful in domains like healthcare, where model interpretability is critical.

**Random Forest** demonstrates strong performance even with datasets containing a mix of numerical and categorical variables and less sensitive to overfitting. Its ability to automatically capture complex interactions between variables makes it a well-suited model for analyzing this dataset.

Overall, Random Forest (with n\_estimators=200, max\_depth=20) is considered the most appropriate choice in practical medical prediction settings.

Therefore, we selected Random Forest with a depth limit— which achieved a higher ROC AUC score— as our final model combination.



Among the three classification models evaluated in this project—Logistic Regression, K-Nearest Neighbors (KNN), and Random Forest—the **Random Forest** model demonstrated the best overall performance. It achieved an Accuracy of approximately **93%** and a **ROC AUC** of over **0.98**, outperforming the other models across all key metrics. Notably, it also showed a high **recall of 0.95** for the minority class (stroke patients labeled as 1).

The KNN model also delivered competitive performance, with a recall of 0.95 and an F1-score of 0.90, but it presents limitations in terms of computational cost.  
 In contrast, Logistic Regression showed relatively lower performance but remains valuable for its interpretability, which is especially important in healthcare-related applications.

**6. Conclusions and recommendations**

It is identified to predict the probability of the stroke only with basic information of patients. Also, health-care data is seriously class imbalanced, including SMOTE, data pre-processes are required.

This model can be useful for the early identification of high-risk individuals for stroke based on initial health screening data collected at public health centers or hospitals.

Since this dataset does not include key features such as blood pressure, it would be beneficial to collect and integrate more complex data such as additional biometric signals and genetic information to improve prediction accuracy. However, ethical considerations must be carefully discussed before collecting patients’ medical data.

**7.** **Appendix**

**Check our full code process in Github:** [**GitHub - flying-adventure/DataScience\_StrokeRiskPrediction**](https://github.com/flying-adventure/DataScience_StrokeRiskPrediction)

Explanation of Advanced Tools and Functions

1. importlib.util

Used to dynamically import Python files as modules during runtime. This allows model scripts to define their own get\_model() function and be loaded into the main pipeline without hardcoding model logic.

spec\_from\_file\_location(name, path): Creates a module spec.

module\_from\_spec(spec): Instantiates module object.

spec.loader.exec\_module(module): Executes module’s code.

2. SMOTE (Synthetic Minority Over-sampling Technique)

Library: imblearn.over\_sampling.SMOTE

Generates synthetic samples of the minority class (stroke=1) to balance the dataset. It uses k-nearest neighbor interpolation.

random\_state=42: ensures reproducibility

Applied only on training data inside cross-validation

3. silhouette\_score

Evaluates how well each point fits into its assigned cluster compared to other clusters. Used to determine optimal number of clusters K in KMeans.

4. PCA (Principal Component Analysis)

Library: sklearn.decomposition.PCA

Used to reduce high-dimensional feature space to 2D for visualization of clusters. It captures the directions (principal components) of maximum variance.

n\_components=2: reduce data to two principal axes for plotting.

5. StratifiedKFold

Ensures that each fold used in cross-validation preserves the percentage of samples for each class label.

n\_splits=8: number of folds

shuffle=True, random\_state=42: controls reproducibility

6. predict\_proba()

Returns class probabilities instead of labels. The second column (index 1) gives the probability of the positive class (stroke=1), enabling risk-based decisions.

7. pd.get\_dummies()

Transforms categorical variables into binary indicator columns (one-hot encoding).

drop\_first=True: avoids dummy variable trap by removing one category per feature.

8. StandardScaler

Standardizes features by removing the mean and scaling to unit variance. Improves convergence in many ML algorithms.

|  |
| --- |
| **FinalMain.py(activate in here)** |
| import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns  from sklearn.preprocessing import StandardScaler  from imblearn.over\_sampling import SMOTE  from sklearn.metrics import accuracy\_score, roc\_auc\_score, classification\_report  from sklearn.model\_selection import StratifiedKFold  import importlib.util  import os  import numpy as np  from sklearn.cluster import KMeans  from sklearn.decomposition import PCA  from sklearn.metrics import silhouette\_score  # List of model file paths  MODEL\_FILES = [  r"knn.py",  r"LogisticRegression\_ver1.py",  r"LogisticRegression\_ver2.py",  r"RandomForest\_ver1.py",  r"RandomForest\_ver2.py"  ]  # Dynamically load models from Python files using get\_model() function  def load\_models\_from\_files():  models = {}  for filepath in MODEL\_FILES:  model\_name = os.path.basename(filepath).split('.')[0]  spec = importlib.util.spec\_from\_file\_location(model\_name, filepath)  module = importlib.util.module\_from\_spec(spec)  spec.loader.exec\_module(module)  models[model\_name] = module.get\_model() # Load model instance  return models  # Data preprocessing function with optional visualization  def preprocess\_data(df, plot=False):  df = df.copy()  if plot:  # 1. Display missing values per column  missing\_values = df.isnull().sum()  # 2. Display class distribution for target label 'stroke'  stroke\_distribution = df['stroke'].value\_counts(normalize=True) \* 100  # 3. Visualize potential outliers  plt.figure(figsize=(15, 5))  plt.subplot(1, 2, 1)  sns.scatterplot(data=df, x='age', y='bmi', alpha=0.5)  plt.title('Scatter Plot: Age vs BMI')  plt.subplot(1, 2, 2)  sns.scatterplot(data=df, x='age', y='avg\_glucose\_level', alpha=0.5)  plt.title('Scatter Plot: Age vs Avg Glucose Level')  plt.tight\_layout()  plt.show()  print(missing\_values)  print(stroke\_distribution)  # Fill missing BMI values with median  df['bmi'] = df['bmi'].fillna(df['bmi'].median())  # Remove rows with undefined gender  df = df[df['gender'] != 'Other']  # Keep adults (age >= 20)  df = df[df['age'] >= 20]  # Remove outliers with extreme BMI values  df = df[df['bmi'] < 80]  # One-hot encode categorical variables  categorical\_cols = ['gender', 'ever\_married', 'Residence\_type', 'work\_type', 'smoking\_status']  df\_encoded = pd.get\_dummies(df, columns=categorical\_cols, drop\_first=True, dtype=int)  X = df\_encoded.drop(columns=['stroke', 'id'])  y = df\_encoded['stroke']  return X, y  # KMeans clustering with silhouette score and PCA visualization  def run\_clustering\_analysis(X\_scaled, y\_res):  best\_k, best\_score = 0, -1  for k in range(2, 11):  kmeans = KMeans(n\_clusters=k, random\_state=42, n\_init=10)  labels = kmeans.fit\_predict(X\_scaled)  score = silhouette\_score(X\_scaled, labels)  if score > best\_score:  best\_k, best\_score = k, score  print(f"\n[Clustering] Best K: {best\_k}, Silhouette Score: {best\_score:.4f}")  # Final clustering with best\_k  kmeans = KMeans(n\_clusters=best\_k, random\_state=42, n\_init=10)  cluster\_labels = kmeans.fit\_predict(X\_scaled)  # Reduce dimensions using PCA for visualization  pca = PCA(n\_components=2)  X\_pca = pca.fit\_transform(X\_scaled)  pca\_df = pd.DataFrame(X\_pca, columns=['PC1', 'PC2'])  pca\_df['cluster'] = cluster\_labels  # Visualize clusters  sns.scatterplot(data=pca\_df, x='PC1', y='PC2', hue='cluster', palette='Set2')  plt.title('PCA Clustering Visualization')  plt.tight\_layout()  plt.show()  # Stroke distribution by cluster  X\_clustered = pd.DataFrame(X\_scaled, columns=X.columns)  X\_clustered['cluster\_label'] = cluster\_labels  df\_cluster = X\_clustered.copy()  df\_cluster['stroke'] = y\_res.reset\_index(drop=True)  print("\n[Stroke ratio statistics by cluster]")  print(df\_cluster.groupby('cluster\_label')['stroke'].value\_counts(normalize=True).unstack().fillna(0))  return X\_clustered  # Evaluate model performance using Stratified K-Fold Cross-Validation  def cross\_validate\_model(model, X, y, k):  skf = StratifiedKFold(n\_splits=k, shuffle=True, random\_state=42)  acc\_scores = []  auc\_scores = []  fold = 1  for train\_idx, test\_idx in skf.split(X, y):  print(f"Evaluating Fold {fold}...")  X\_train, X\_test = X.iloc[train\_idx], X.iloc[test\_idx]  y\_train, y\_test = y.iloc[train\_idx], y.iloc[test\_idx]  smote = SMOTE(random\_state=42)  X\_train\_res, y\_train\_res = smote.fit\_resample(X\_train, y\_train) # Apply SMOTE only to training set  scaler = StandardScaler()  X\_train\_scaled = scaler.fit\_transform(X\_train\_res)  X\_test\_scaled = scaler.transform(X\_test)  model.fit(X\_train\_scaled, y\_train\_res)  y\_pred = model.predict(X\_test\_scaled)  y\_prob = model.predict\_proba(X\_test\_scaled)[:, 1]  acc\_scores.append(accuracy\_score(y\_test, y\_pred))  auc\_scores.append(roc\_auc\_score(y\_test, y\_prob))  fold += 1  return np.mean(acc\_scores), np.mean(auc\_scores)  # Plot bar chart comparing model performance  def plot\_model\_comparison(results):  df\_plot = pd.DataFrame(results).T  df\_plot = df\_plot[['accuracy', 'roc\_auc']]  df\_plot.plot(kind='bar', figsize=(10,6))  plt.title('Model Performance Comparison')  plt.ylabel('Score')  plt.xticks(rotation=45)  plt.ylim(0, 1)  plt.tight\_layout()  plt.show()  # Process input patient data and format as encoded DataFrame for prediction  def preprocess\_new\_data(new\_data\_list, X\_columns):  df = pd.DataFrame(new\_data\_list)  df['bmi'] = df['bmi'].fillna(df['bmi'].median())  df = df[df['age'] >= 20]  df = df[df['bmi'] < 80]  categorical\_cols = ['gender', 'ever\_married', 'Residence\_type', 'work\_type', 'smoking\_status']  df\_encoded = pd.get\_dummies(df, columns=categorical\_cols, drop\_first=True, dtype=int)  for col in X\_columns:  if col not in df\_encoded.columns:  df\_encoded[col] = 0  df\_encoded = df\_encoded[X\_columns]  return df\_encoded  # Select the best model based on a weighted scoring system  # The default weights prioritize ROC AUC (60%) over Accuracy (40%)  def select\_best\_model(results, weights={'roc\_auc': 0.6, 'accuracy': 0.4}):  scores = {}  for name, res in results.items():  # Compute weighted score for each model  score = (res['roc\_auc'] \* weights['roc\_auc'] + res['accuracy'] \* weights['accuracy'])  scores[name] = score  # Return model with the highest overall score  return max(scores.items(), key=lambda x: x[1])  # Function to collect patient information from the user through CLI inputs  # Ensures data integrity and valid ranges for each input  # Returns a list of dictionaries, each representing one patient  def input\_patient\_data():  patients = []  num\_patients = None  while num\_patients is None:  try:  num\_patients = int(input("Enter the number of patients to predict: "))  if num\_patients <= 0:  print("You must enter at least 1 patient.")  num\_patients = None  except ValueError:  print("Please enter a number.")  for i in range(num\_patients):  print(f"\nEnter information for patient {i+1}:")  # Gender  while True:  gender = input("Gender (Male/Female): ").strip()  if gender in ['Male', 'Female']:  break  print("Gender must be either 'Male' or 'Female'.")  # Age  while True:  try:  age = float(input("Age: "))  if 20 <= age <= 120:  break  else:  print("Age must be between 20 and 120.")  except ValueError:  print("Please enter a number.")  # Hypertension  while True:  try:  hypertension = int(input("Hypertension (0: No, 1: Yes): "))  if hypertension in [0, 1]:  break  else:  print("Only 0 or 1 is allowed.")  except ValueError:  print("Please enter a number.")  # Heart disease  while True:  try:  heart\_disease = int(input("Heart disease (0: No, 1: Yes): "))  if heart\_disease in [0, 1]:  break  else:  print("Only 0 or 1 is allowed.")  except ValueError:  print("Please enter a number.")  # Average glucose level  while True:  try:  avg\_glucose\_level = float(input("Average glucose level: "))  if 0 <= avg\_glucose\_level <= 500:  break  else:  print("Glucose level must be between 0 and 500.")  except ValueError:  print("Please enter a number.")  # BMI  while True:  try:  bmi = float(input("BMI: "))  if 10 <= bmi < 80:  break  else:  print("BMI must be between 10 and 80.")  except ValueError:  print("Please enter a number.")  # Work type selection  work\_types = ['Private', 'Self-employed', 'Govt\_job', 'children', 'Never\_worked']  while True:  work\_type = input("Work type (Private/Self-employed/Govt\_job/children/Never\_worked): ").strip()  if work\_type in work\_types:  break  print(f"Work type must be one of: {', '.join(work\_types)}.")  # Smoking status  smoking\_statuses = ['never smoked', 'formerly smoked', 'smokes', 'Unknown']  while True:  smoking\_status = input("Smoking status (never smoked/formerly smoked/smokes/Unknown): ").strip()  if smoking\_status in smoking\_statuses:  break  print(f"Smoking status must be one of: {', '.join(smoking\_statuses)}.")  # Residence type  while True:  residence\_type = input("Residence Type (Urban/Rural): ").strip()  if residence\_type in ['Urban', 'Rural']:  break  print("Residence type must be either 'Urban' or 'Rural'.")  # Marital status  while True:  ever\_married = input("Marital Status (Yes/No): ").strip()  if ever\_married in ['Yes', 'No']:  break  print("Marital status must be 'Yes' or 'No'.")  # Aggregate patient input into dictionary format  patient = {  'gender': gender,  'age': age,  'hypertension': hypertension,  'heart\_disease': heart\_disease,  'avg\_glucose\_level': avg\_glucose\_level,  'bmi': bmi,  'work\_type': work\_type,  'smoking\_status': smoking\_status,  'Residence\_type': residence\_type,  'ever\_married': ever\_married  }  patients.append(patient)  return patients  # Running Main Function  if \_\_name\_\_ == "\_\_main\_\_":  # Load Model  models = load\_models\_from\_files()  # Load Data  df = pd.read\_csv("healthcare-dataset-stroke-data.csv")  # Dynamic import and function call of kfoldcross.py (only finding optimal k here)  kfoldcross\_path = r"kfoldcross.py"  spec = importlib.util.spec\_from\_file\_location("kfoldcross", kfoldcross\_path)  kfoldcross = importlib.util.module\_from\_spec(spec)  spec.loader.exec\_module(kfoldcross)  print("▶ Finding Optimal K (K-Fold Cross Validation) in progress...")  X, y = preprocess\_data(df, plot=True)  best\_k, k\_auc\_dict = kfoldcross.find\_best\_k\_and\_evaluate(X, y)  print(f"Optional K: {best\_k}")  print(f"AUC values per K: {k\_auc\_dict}")  # Cross-validation per model  results = {}  for name, model in models.items():  print(f"\n▶ Model: {name}")  acc, auc = cross\_validate\_model(model, X, y, best\_k)  results[name] = {'accuracy': acc, 'roc\_auc': auc}  print(f"[{name}] Average Accuracy: {acc:.4f}, Average ROC AUC: {auc:.4f}")  # Output Final Results  print("\nFinal Evaluation Results per Model")  for name, res in results.items():  print(f"<{name}>")  print(f"Accuracy: {res['accuracy']:.4f}, ROC AUC: {res['roc\_auc']:.4f}")    plot\_model\_comparison(results)    # Select Best Performing Model  best\_model\_name, best\_score = select\_best\_model(results)  best\_model = models[best\_model\_name]  print(f"\n=== Final Selected Model: {best\_model\_name} (Overall Score: {best\_score:.4f}) ===")    # Re-train on the entire dataset and define the scaler  print("\n=== Re-train Final Model ===")  X, y = preprocess\_data(df, plot=False)    # Apply SMOTE (entire dataset)  smote = SMOTE(random\_state=42)  X\_res, y\_res = smote.fit\_resample(X, y)    # Re-define and train the scaler  global\_scaler = StandardScaler()  X\_scaled = global\_scaler.fit\_transform(X\_res)    # Re-train the final model  best\_model.fit(X\_scaled, y\_res)  # Run clustering analysis (KMeans + PCA)  X\_clustered = run\_clustering\_analysis(X\_scaled, y\_res)  # Receive patient data as user input  print("\n=== Input new patient information ===")  new\_patients\_data = input\_patient\_data()  # Preprocess prediction data  X\_new = preprocess\_new\_data(new\_patients\_data, X.columns)  X\_new\_scaled = global\_scaler.transform(X\_new)  # Predict and output results  probs = best\_model.predict\_proba(X\_new\_scaled)[:, 1]  for i, prob in enumerate(probs):  print(f"\n--- Patient {i+1} ---")  print(f"Input data: {new\_patients\_data[i]}")  print(f"Predicted stroke probability: {prob:.4f}")  if prob >= 0.5:  print("Predicts a high likelihood of stroke.")  else:  print("Predicts a low likelihood of stroke.") |

|  |
| --- |
| **knn.py** |
| from sklearn.neighbors import KNeighborsClassifier  def get\_model():  return KNeighborsClassifier(  n\_neighbors=13,  weights='uniform',  p=2  ) |

|  |
| --- |
| **LogisticRegression\_ver1.py** |
| from sklearn.linear\_model import LogisticRegression  def get\_model():  return LogisticRegression(  C=1,  penalty='l2',  solver='liblinear',  random\_state=42  ) |

|  |
| --- |
| **LogisticRegression\_ver2.py** |
| from sklearn.linear\_model import LogisticRegression  def get\_model():  return LogisticRegression(  C=0.1,  penalty='l1',  solver='liblinear',  random\_state=42  ) |

|  |
| --- |
| **RandomForest\_ver1.py** |
| from sklearn.ensemble import RandomForestClassifier  def get\_model():  return RandomForestClassifier(  n\_estimators=100,  max\_depth=None,  random\_state=42  ) |

|  |
| --- |
| **RandomForest\_ver2.py** |
| from sklearn.ensemble import RandomForestClassifier  def get\_model():  return RandomForestClassifier(  n\_estimators=200,  max\_depth=20,  random\_state=42  ) |

|  |
| --- |
| **kfoldcross.py (K-Fold Cross Validation model to validate above models)** |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns  from sklearn.ensemble import RandomForestClassifier  from sklearn.model\_selection import StratifiedKFold  from sklearn.metrics import roc\_auc\_score, classification\_report  from imblearn.over\_sampling import SMOTE  def show\_feature\_importance(X, y):  model\_feat = RandomForestClassifier(random\_state=42)  model\_feat.fit(X, y)  feat\_importances = pd.Series(model\_feat.feature\_importances\_, index=X.columns)  print("\n[Feature Importance Analysis]")  print("Top 10 Feature Importances:")  print(feat\_importances.nlargest(10))  plt.figure(figsize=(10, 6))  feat\_importances.nlargest(10).plot(kind='barh')  plt.title("Top Feature Importances")  plt.tight\_layout()  plt.show()  def find\_best\_k\_and\_evaluate(X, y, k\_values=range(5, 11)):  show\_feature\_importance(X, y)  best\_k = None  best\_score = 0  all\_scores = {}  print("▶ Performing Stratified K-Fold + SMOTE + RandomForestClassifier for each K value...")  for k in k\_values:  print(f"\n=== K={k} ===")  skf = StratifiedKFold(n\_splits=k, shuffle=True, random\_state=42)  fold\_auc\_scores = []  fold = 1  for train\_idx, test\_idx in skf.split(X, y):  print(f"Fold {fold} in progress...")  X\_train, X\_test = X.iloc[train\_idx], X.iloc[test\_idx]  y\_train, y\_test = y.iloc[train\_idx], y.iloc[test\_idx]  smote = SMOTE(random\_state=42)  X\_train\_res, y\_train\_res = smote.fit\_resample(X\_train, y\_train)  model = RandomForestClassifier(random\_state=42, n\_estimators=100)  model.fit(X\_train\_res, y\_train\_res)  y\_prob = model.predict\_proba(X\_test)[:, 1]  auc\_score = roc\_auc\_score(y\_test, y\_prob)  fold\_auc\_scores.append(auc\_score)  print(f"Fold {fold} ROC AUC: {auc\_score:.4f}")  fold += 1  mean\_auc = np.mean(fold\_auc\_scores)  all\_scores[k] = mean\_auc  print(f"K={k} Average ROC AUC: {mean\_auc:.4f}")  if mean\_auc > best\_score:  best\_score = mean\_auc  best\_k = k  print(f"\n✅ The best K is {best\_k}, and the average ROC AUC is {best\_score:.4f}")  return best\_k, all\_scores |

**8.** **Reference**

* *Kaggle Stroke data set* https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset
* Wang, S. and Yao, X. (2009). Diversity analysis on imbalanced data sets by using ensemble models. 2009 IEEE Symposium on Computational Intelligence and Data Mining, 324-331.
* <https://doi.org/10.1109/cidm.2009.4938667>
* [Example of cross-validation with unbalanced data · GitHub](https://gist.github.com/kiwidamien/bcbe8e527a5f0cc9f28c4fe692f70cbc)
* [Unbalanced data and cross-validation | Kaggle](https://www.kaggle.com/discussions/questions-and-answers/27589)
* [How to Fix k-Fold Cross-Validation for Imbalanced Classification -](https://machinelearningmastery.com/cross-validation-for-imbalanced-classification/) [MachineLearningMastery.com](http://machinelearningmastery.com)
* [machine learning - How to perform SMOTE with cross validation in sklearn in python - Stack Overflow](https://stackoverflow.com/questions/55591063/how-to-perform-smote-with-cross-validation-in-sklearn-in-python)
* [Imbalanced-Data-with-SMOTE-Techniques/Imbalanced\_Data.ipynb at main · Swastik-25/Imbalanced-Data-with-SMOTE-Techniques · GitHub](https://github.com/Swastik-25/Imbalanced-Data-with-SMOTE-Techniques/blob/main/Imbalanced_Data.ipynb)