

Stroke Dataset - Machine Learning Assignment

To Dr:Hager Saleh

Student Name: Hussam Almokayed

```
In [1]: # =====  
# Libraries & Environment Setup  
# =====  
  
import warnings  
warnings.filterwarnings("ignore")  
  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from imblearn.pipeline import Pipeline as ImbPipeline  
from imblearn.over_sampling import SMOTE  
  
from sklearn.impute import SimpleImputer  
from sklearn.model_selection import train_test_split, GridSearchCV  
from sklearn.pipeline import Pipeline  
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder  
  
from sklearn.feature_selection import SelectKBest, chi2, RFE  
  
from sklearn.linear_model import LogisticRegression  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier  
from sklearn.svm import SVC  
from sklearn.naive_bayes import GaussianNB  
from sklearn.neighbors import KNeighborsClassifier  
  
from sklearn.metrics import (  
    confusion_matrix,  
    accuracy_score,  
    precision_score,  
    recall_score,  
    f1_score,  
    classification_report  
)
```

```

from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTETomek

from xgboost import XGBClassifier

```

```

In [2]: from xgboost import XGBClassifier
print("XGBoost is ready ✅")

```

XGBoost is ready ✅

```

In [3]: pwd

```

```

Out[3]: 'C:\\Users\\Eng-Hussam\\Desktop\\الواجب الثاني من الدكتور هاجر\\Master Ai-Folder'

```

```

In [4]: # 1-Download data set and printed
df = pd.read_csv("healthcare-dataset-stroke-data.csv")
df.head()

```

```

Out[4]:

```

	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

```

In [5]: df.columns

```

```

Out[5]: Index(['id', 'gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
              'work_type', 'Residence_type', 'avg_glucose_level', 'bmi',
              'smoking_status', 'stroke'],
              dtype='object')

```

```

In [6]: # remove ID from Data Set
df.drop(columns=["id"], inplace=True)

```

```

In [7]: df['stroke'].value_counts()

```

```
Out[7]: stroke
0      4861
1       249
Name: count, dtype: int64
```

```
In [8]: # to know the data type
df.dtypes
```

```
Out[8]: gender      object
age      float64
hypertension    int64
heart_disease   int64
ever_married    object
work_type      object
Residence_type  object
avg_glucose_level float64
bmi            float64
smoking_status  object
stroke         int64
dtype: object
```

```
In [9]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  -
0   gender                 5110 non-null   object
1   age                   5110 non-null   float64
2   hypertension           5110 non-null   int64
3   heart_disease          5110 non-null   int64
4   ever_married           5110 non-null   object
5   work_type              5110 non-null   object
6   Residence_type         5110 non-null   object
7   avg_glucose_level      5110 non-null   float64
8   bmi                   4909 non-null   float64
9   smoking_status         5110 non-null   object
10  stroke                 5110 non-null   int64
dtypes: float64(3), int64(3), object(5)
memory usage: 439.3+ KB
```

```
In [10]: # to know the missing value
df.isnull().sum()
```

```
Out[10]: 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
```

```
In [11]: # 2- Handle missing values appropriately BMI لضمان عدم فقدان بيانات مهمة BMI استخدمنا المتوسط لتعويض القيم المفقودة في BMI
# معالجة القيم المفقودة باستخدام المتوسط للعمود العددي الوحيد الذي به نقص (bmi)
imputer = SimpleImputer(strategy='mean')
df['bmi'] = imputer.fit_transform(df[['bmi']])
```

```
In [12]: # Check missing values after
print(df.isnull().sum())
```

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

```
In [13]: df.describe()
```

Out[13]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000
mean	43.226614	0.097456	0.054012	106.147677	28.893237	0.048728
std	22.612647	0.296607	0.226063	45.283560	7.698018	0.215320
min	0.080000	0.000000	0.000000	55.120000	10.300000	0.000000
25%	25.000000	0.000000	0.000000	77.245000	23.800000	0.000000
50%	45.000000	0.000000	0.000000	91.885000	28.400000	0.000000
75%	61.000000	0.000000	0.000000	114.090000	32.800000	0.000000
max	82.000000	1.000000	1.000000	271.740000	97.600000	1.000000

In [14]:

```
df.head()
```

Out[14]:

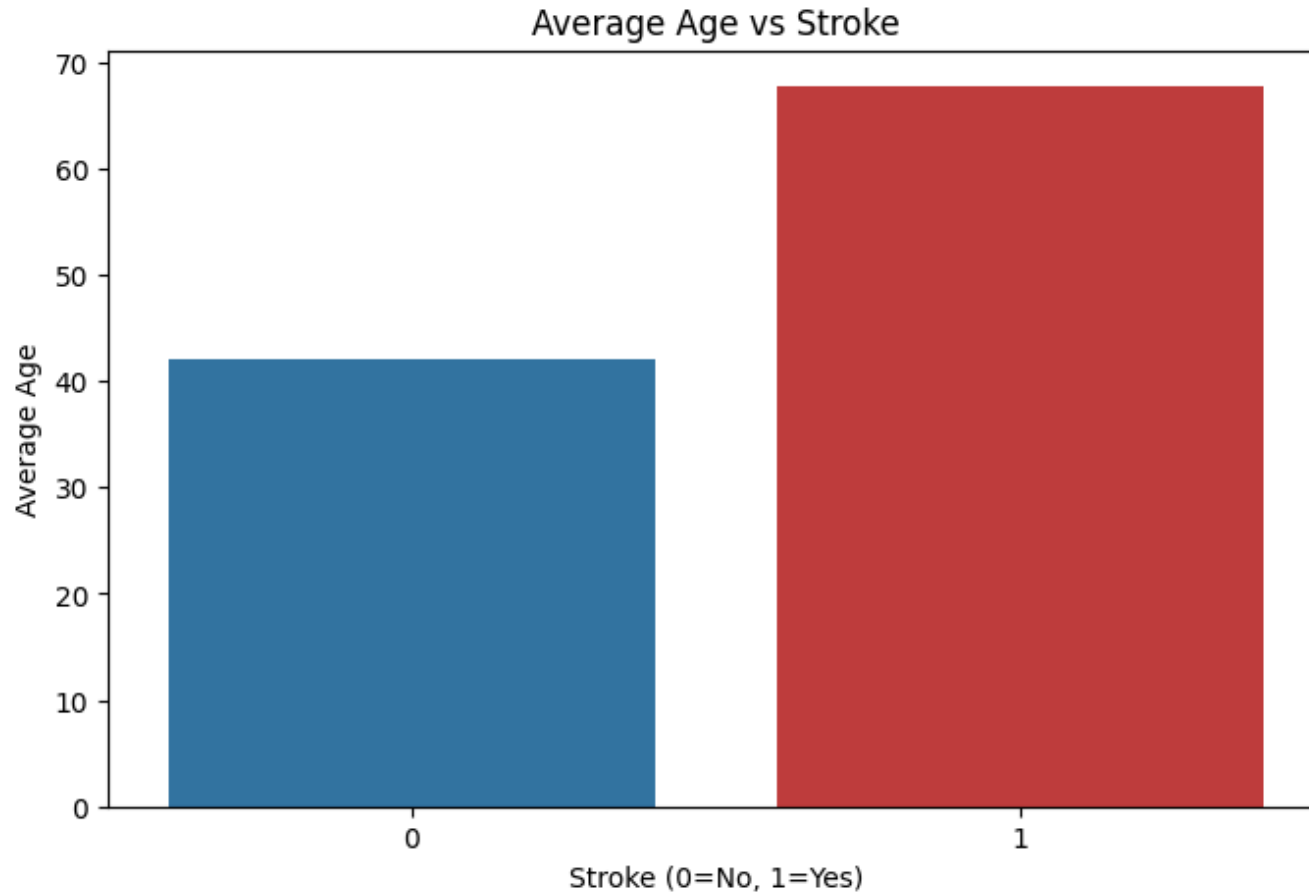
	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	Male	67.0	0	1	Yes	Private	Urban	228.69	36.600000	formerly smoked	1
1	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	28.893237	never smoked	1
2	Male	80.0	0	1	Yes	Private	Rural	105.92	32.500000	never smoked	1
3	Female	49.0	0	0	Yes	Private	Urban	171.23	34.400000	smokes	1
4	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.000000	never smoked	1

In [15]:

```
# Print Plt
plt.figure(figsize=(8,5))

sns.barplot(
    data=df,
    x="stroke",
    y="age",
    estimator="mean",
    errorbar=None,
    palette=["#1f77b4", "#d62728"]
)
```

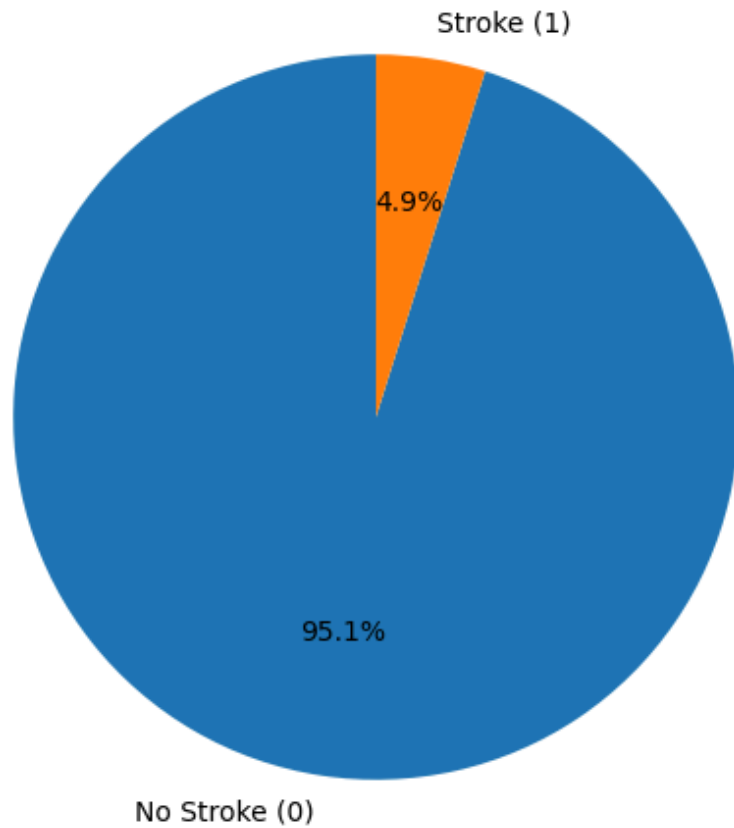
```
plt.title("Average Age vs Stroke")
plt.xlabel("Stroke (0=No, 1=Yes)")
plt.ylabel("Average Age")
plt.show()
```



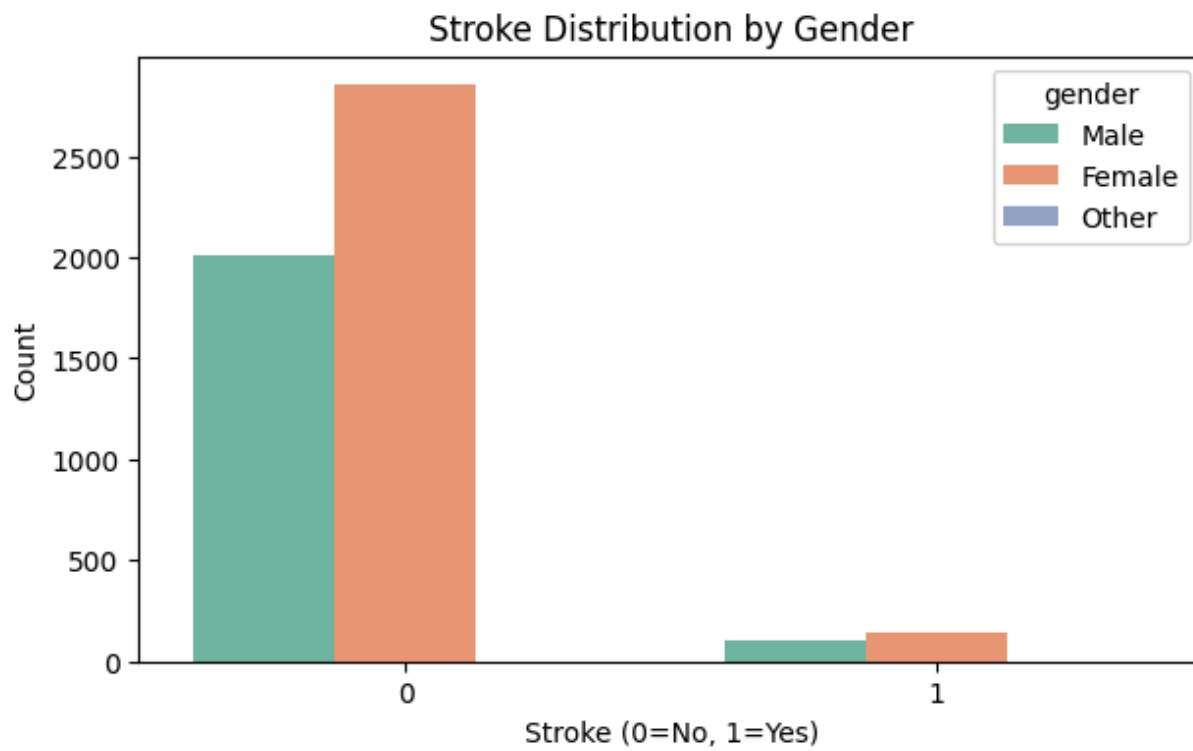
```
In [16]: print(df["stroke"].value_counts())
stroke_counts = df["stroke"].value_counts()
plt.figure(figsize=(6,6))
plt.pie(
stroke_counts,
labels=["No Stroke (0)", "Stroke (1)"],
autopct="%1.1f%%",
startangle=90
)
plt.title("Distribution of Stroke Classes")
plt.show()
```

```
stroke
0    4861
1     249
Name: count, dtype: int64
```

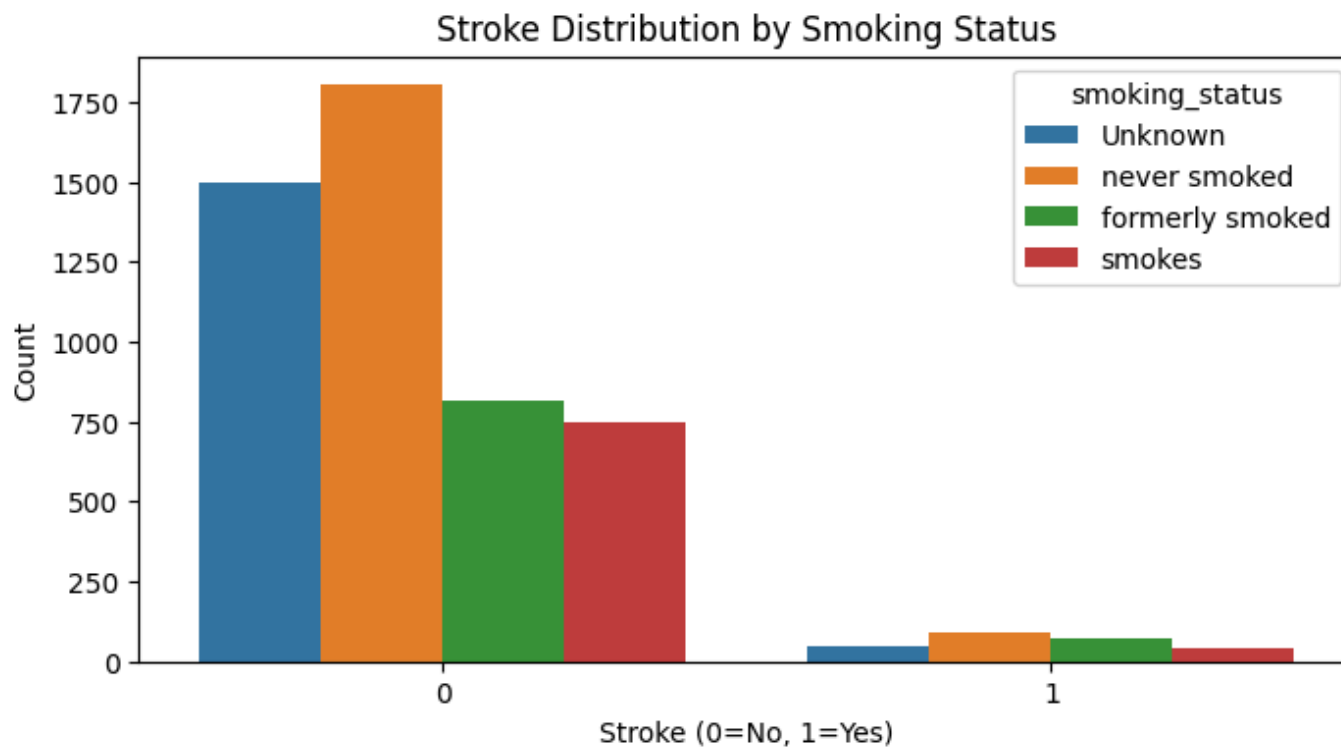
Distribution of Stroke Classes



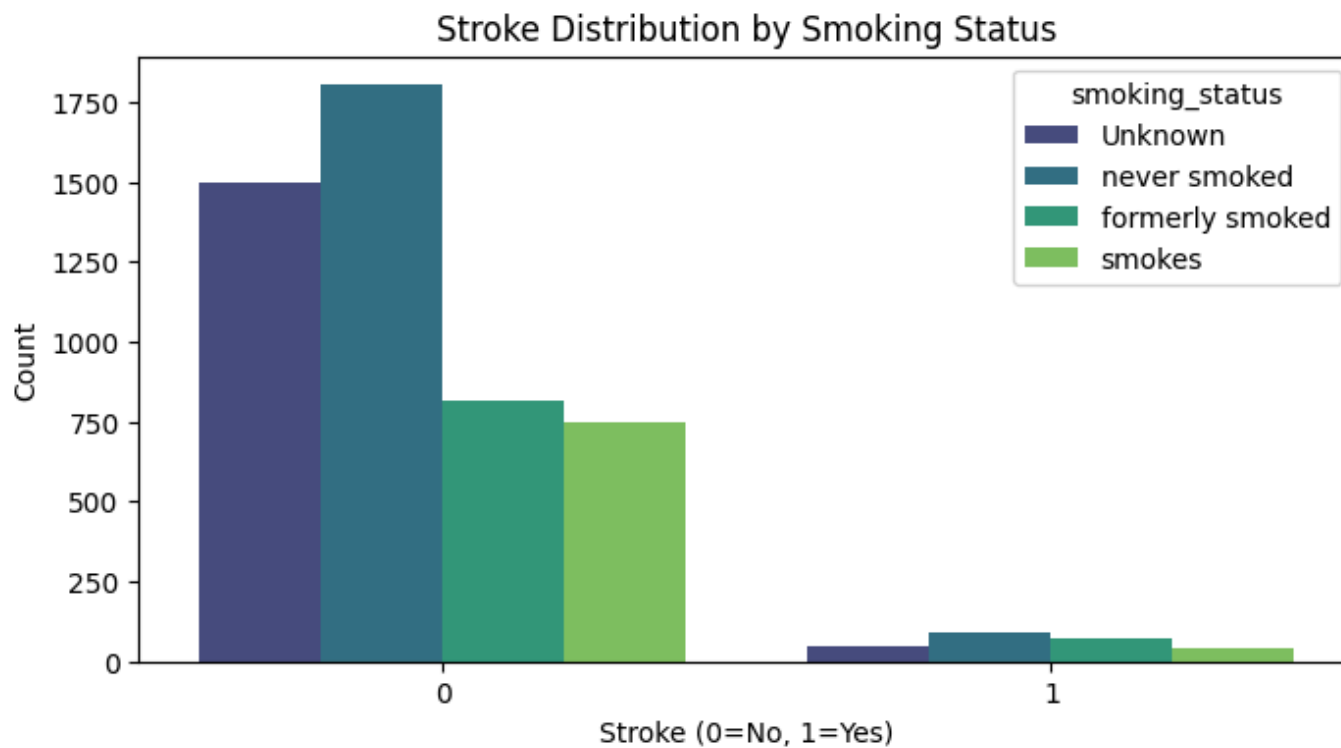
```
In [17]: plt.figure(figsize=(7,4))
sns.countplot(
    data=df,
    x="stroke",
    hue="gender",
    palette="Set2"
)
plt.title("Stroke Distribution by Gender")
plt.xlabel("Stroke (0=No, 1=Yes)")
plt.ylabel("Count")
plt.show()
```



```
In [18]: plt.figure(figsize=(8,4))
sns.countplot(data=df, x="stroke", hue="smoking_status")
plt.title("Stroke Distribution by Smoking Status")
plt.xlabel("Stroke (0=No, 1=Yes)")
plt.ylabel("Count")
plt.show()
```



```
In [19]: plt.figure(figsize=(8,4))
sns.countplot(
    data=df,
    x="stroke",
    hue="smoking_status",
    palette="viridis"
)
plt.title("Stroke Distribution by Smoking Status")
plt.xlabel("Stroke (0=No, 1=Yes)")
plt.ylabel("Count")
plt.show()
```

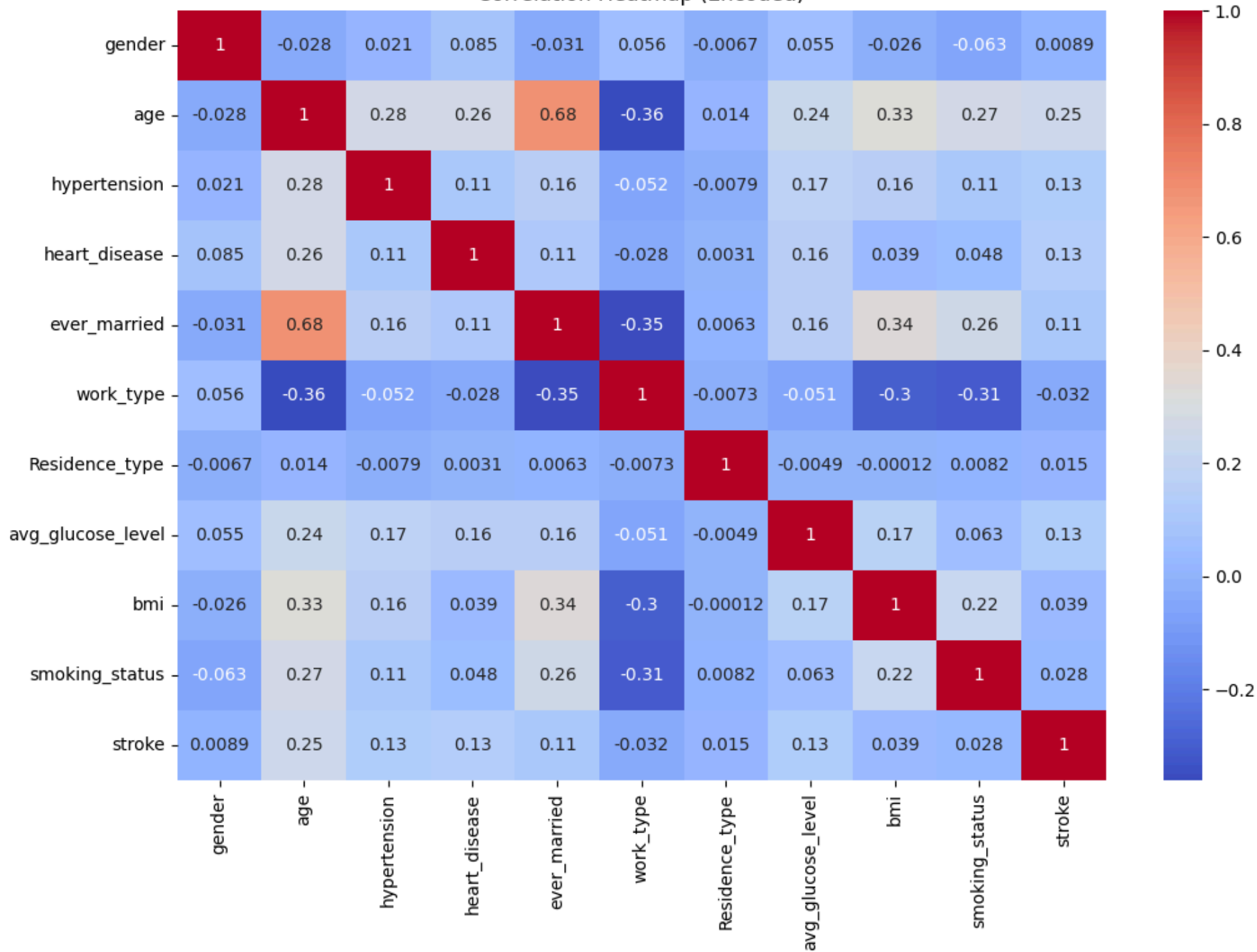


In [20]: *# encoding then draw the Plt Figure*

```
df_encoded = df.copy()
for col in df_encoded.select_dtypes(include=["object"]).columns:
    df_encoded[col] = LabelEncoder().fit_transform(df_encoded[col].astype(str))

plt.figure(figsize=(12,8))
sns.heatmap(df_encoded.corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap (Encoded)")
plt.show()
```

Correlation Heatmap (Encoded)



```
In [21]: # Printed to check if still categorical data
df.head()
```

```
Out[21]:
```

	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	36.600000	formerly smoked	1
1	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	28.893237	never smoked	1
2	Male	80.0	0	1	Yes	Private	Rural	105.92	32.500000	never smoked	1
3	Female	49.0	0	0	Yes	Private	Urban	171.23	34.400000	smokes	1
4	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.000000	never smoked	1

```
In [22]: # Labeling the data set before training تحويل البيانات النصية إلى أرقام قبل
label_encoders = {}

for column in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le
```

```
In [23]: df.head()
```

```
Out[23]:
```

	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	36.600000	1	1
1	0	61.0	0	0	1	3	0	202.21	28.893237	2	1
2	1	80.0	0	1	1	2	0	105.92	32.500000	2	1
3	0	49.0	0	0	1	2	1	171.23	34.400000	3	1
4	0	79.0	1	0	1	3	0	174.12	24.000000	2	1

```
In [24]: df['stroke'].value_counts()
```

```
Out[24]: stroke
0      4861
1       249
Name: count, dtype: int64
```

Imbalance dataset

```
In [25]: # Define features and target
X = df.drop(columns=["stroke"])
y = df["stroke"]
```

```
In [26]: X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)
```

i use SMOTE + Tomek method Links on training data only

```
In [27]: from imblearn.combine import SMOTETomek

# Apply SMOTE + Tomek Links on training data only
smt = SMOTETomek(random_state=42)
X_smt, y_smt = smt.fit_resample(X_train, y_train)

# Check class distribution after resampling
print("After SMOTE + Tomek Links:")
print(pd.Series(y_smt).value_counts())
```

After SMOTE + Tomek Links:

```
stroke
0      3865
1      3865
Name: count, dtype: int64
```

i apply Apply machine learning models with GridSearchCV + Evaluate performance (Added: Gradient Boosting + XGBoost)

```
In [28]: # ----- Confusion Matrix (Sky Blue & White) -----
def plot_confusion_matrix(y_true, y_pred, model_name):
    cm = confusion_matrix(y_true, y_pred)
    labels = np.unique(y_true)

    plt.figure(figsize=(4, 4))
    sns.heatmap(
```

```

        cm, annot=True, fmt='d', cmap="Blues", cbar=False,
        linewidths=1, linecolor="white",
        xticklabels=labels, yticklabels=labels
    )
    plt.title(f'Confusion Matrix for {model_name}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.tight_layout()
    plt.show()

# ----- Data -----
X = df.drop(columns=["stroke"])
y = df["stroke"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state=42, stratify=y
)

# ----- XGBoost imbalance -----
pos = (y_train == 1).sum()
neg = (y_train == 0).sum()
scale_pos_weight = (neg / pos) if pos > 0 else 1.0

# ----- Store results to -----
final_results = []
best_models = {}

# ----- GridSearch settings -----
CV_FOLDS = 3
N_JOBS = 1

```

Random Forest Model Trained on SMOTE–Tomek Balanced Training Data

```

In [29]: name = "Random Forest"

model = RandomForestClassifier(random_state=42, class_weight="balanced")
param_grid = {
    "n_estimators": [200],
    "max_depth": [None, 20],
    "min_samples_split": [2, 10],
    "min_samples_leaf": [1, 4],
    "max_features": ["sqrt"]
}

grid = GridSearchCV(model, param_grid, cv=CV_FOLDS, scoring="f1_macro", n_jobs=N_JOBS)

```

```

grid.fit(X_smt, y_smt)

best_models[name] = grid.best_estimator_
y_pred = best_models[name].predict(X_test)

print("Best Params:", grid.best_params_)
print(classification_report(y_test, y_pred, zero_division=0))
plot_confusion_matrix(y_test, y_pred, name)

test_accuracy = accuracy_score(y_test, y_pred)
test_f1_macro = f1_score(y_test, y_pred, average="macro", zero_division=0)
test_recall_class1 = recall_score(y_test, y_pred, pos_label=1, average="binary", zero_division=0)
test_f1_class1 = f1_score(y_test, y_pred, pos_label=1, average="binary", zero_division=0)

final_results.append({
    "Model": name,
    "CV_F1_macro": grid.best_score_,
    "Test_Accuracy": test_accuracy,
    "Test_F1_macro": test_f1_macro,
    "Test_Recall_class1": test_recall_class1,
    "Test_F1_class1": test_f1_class1
})

```

Best Params: {'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

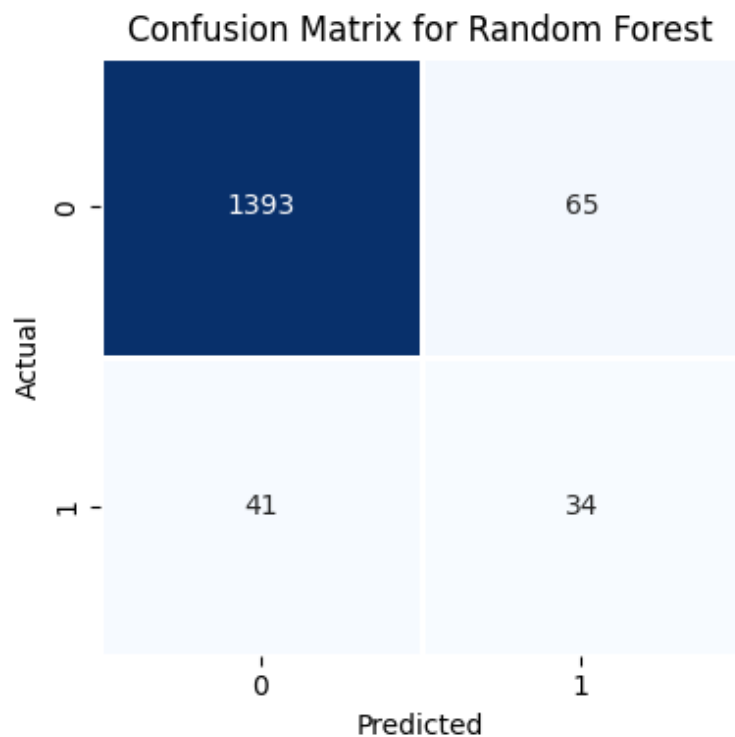
0	0.97	0.96	0.96	1458
---	------	------	------	------

1	0.34	0.45	0.39	75
---	------	------	------	----

accuracy			0.93	1533
----------	--	--	------	------

macro avg	0.66	0.70	0.68	1533
-----------	------	------	------	------

weighted avg	0.94	0.93	0.94	1533
--------------	------	------	------	------



```
In [30]: name = "Logistic Regression (SMOTE)"

model = ImbPipeline([
    ("smote", SMOTE(random_state=42)),
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(max_iter=2000, class_weight="balanced", random_state=42))
])

param_grid = {
    "clf__C": [0.1, 1, 10]
}

grid = GridSearchCV(model, param_grid, cv=CV_FOLDS, scoring="f1_macro", n_jobs=N_JOBS)
grid.fit(X_train, y_train)

best_models[name] = grid.best_estimator_
y_pred = best_models[name].predict(X_test)

print("Best Params:", grid.best_params_)
print(classification_report(y_test, y_pred, zero_division=0))
plot_confusion_matrix(y_test, y_pred, name)
```

```

test_accuracy = accuracy_score(y_test, y_pred)
test_f1_macro = f1_score(y_test, y_pred, average="macro", zero_division=0)
test_recall_class1 = recall_score(y_test, y_pred, pos_label=1, average="binary", zero_division=0)
test_f1_class1 = f1_score(y_test, y_pred, pos_label=1, average="binary", zero_division=0)

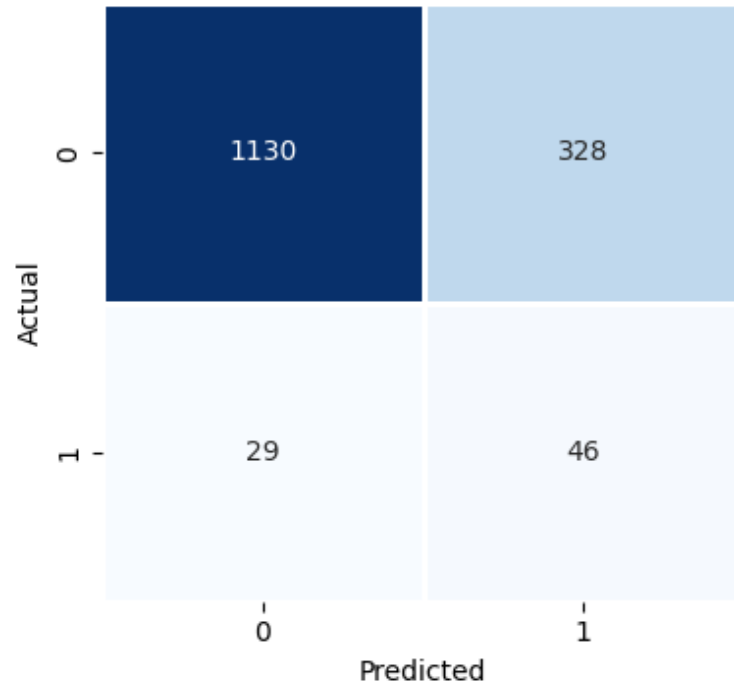
final_results.append({
    "Model": name,
    "CV_F1_macro": grid.best_score_,
    "Test_Accuracy": test_accuracy,
    "Test_F1_macro": test_f1_macro,
    "Test_Recall_class1": test_recall_class1,
    "Test_F1_class1": test_f1_class1
})

```

Best Params: {'clf__C': 1}

	precision	recall	f1-score	support
0	0.97	0.78	0.86	1458
1	0.12	0.61	0.20	75
accuracy			0.77	1533
macro avg	0.55	0.69	0.53	1533
weighted avg	0.93	0.77	0.83	1533

Confusion Matrix for Logistic Regression (SMOTE)



In []:

In [31]:

```
name = "SVM (RBF) (SMOTETomek)"

model = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", SVC(kernel="rbf", class_weight="balanced", probability=True, random_state=42))
])

param_grid = {
    "clf__C": [1, 10],
    "clf__gamma": ["scale", 0.1]
}

grid = GridSearchCV(model, param_grid, cv=CV_FOLDS, scoring="f1_macro", n_jobs=N_JOBS)
grid.fit(X_smt, y_smt) # ✅ تدريب على بيانات متوازنة مسبقًا

best_models[name] = grid.best_estimator_
y_pred = best_models[name].predict(X_test)
```

Gradient Boosting Trained on SMOTE–Tomek Balanced Training Data

```
In [32]: name = "Gradient Boosting"

model = GradientBoostingClassifier(random_state=42)
param_grid = {
    "n_estimators": [100, 200],
    "learning_rate": [0.05, 0.1],
    "max_depth": [2, 3]
}

grid = GridSearchCV(model, param_grid, cv=CV_FOLDS, scoring="f1_macro", n_jobs=N_JOBS)
grid.fit(X_train, y_train)

best_models[name] = grid.best_estimator_
y_pred = best_models[name].predict(X_test)

print("Best Params:", grid.best_params_)
print(classification_report(y_test, y_pred, zero_division=0))
plot_confusion_matrix(y_test, y_pred, name)

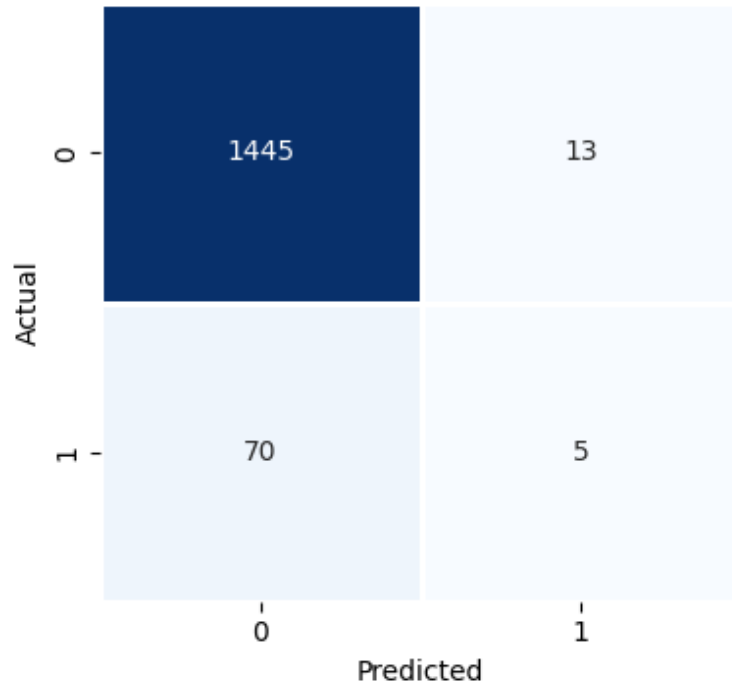
test_accuracy = accuracy_score(y_test, y_pred)
test_f1_macro = f1_score(y_test, y_pred, average="macro", zero_division=0)
test_recall_class1 = recall_score(y_test, y_pred, pos_label=1, average="binary", zero_division=0)
test_f1_class1 = f1_score(y_test, y_pred, pos_label=1, average="binary", zero_division=0)

final_results.append({
    "Model": name,
    "CV_F1_macro": grid.best_score_,
    "Test_Accuracy": test_accuracy,
    "Test_F1_macro": test_f1_macro,
    "Test_Recall_class1": test_recall_class1,
    "Test_F1_class1": test_f1_class1
})
```

Best Params: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200}

	precision	recall	f1-score	support
0	0.95	0.99	0.97	1458
1	0.28	0.07	0.11	75
accuracy			0.95	1533
macro avg	0.62	0.53	0.54	1533
weighted avg	0.92	0.95	0.93	1533

Confusion Matrix for Gradient Boosting



XGBoost with Class Imbalance Weighting (scale_pos_weight) + GridSearchCV

```
In [33]: name = "XGBoost"

model = XGBClassifier(
    random_state=42,
    eval_metric="logloss",
    scale_pos_weight=scale_pos_weight
)

param_grid = {
    "n_estimators": [200],
    "learning_rate": [0.05, 0.1],
    "max_depth": [3, 5],
    "subsample": [0.8],
    "colsample_bytree": [0.8]
}

grid = GridSearchCV(model, param_grid, cv=CV_FOLDS, scoring="f1_macro", n_jobs=N_JOBS)
grid.fit(X_train, y_train)
```

```

best_models[name] = grid.best_estimator_
y_pred = best_models[name].predict(X_test)

print("Best Params:", grid.best_params_)
print(classification_report(y_test, y_pred, zero_division=0))
plot_confusion_matrix(y_test, y_pred, name)

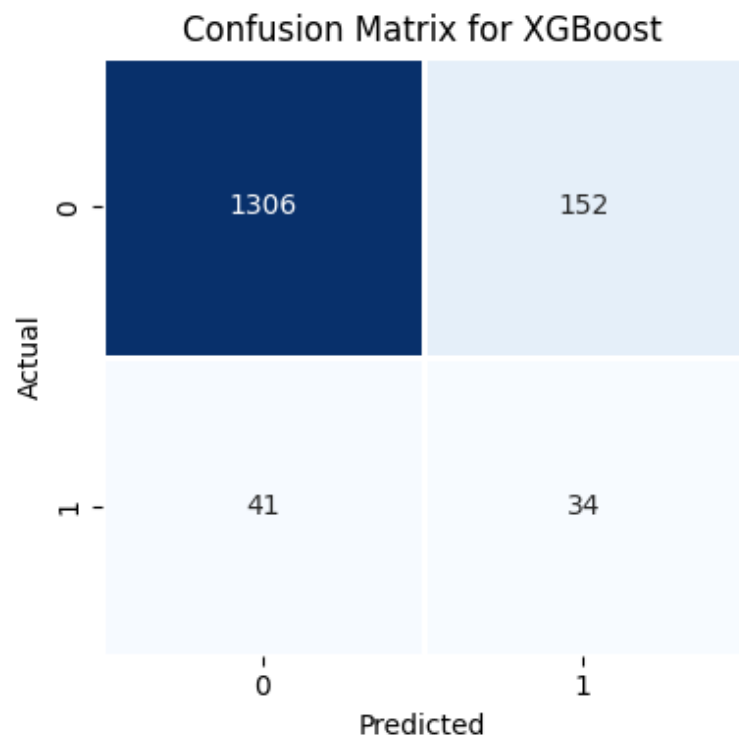
test_accuracy = accuracy_score(y_test, y_pred)
test_f1_macro = f1_score(y_test, y_pred, average="macro", zero_division=0)
test_recall_class1 = recall_score(y_test, y_pred, pos_label=1, average="binary", zero_division=0)
test_f1_class1 = f1_score(y_test, y_pred, pos_label=1, average="binary", zero_division=0)

final_results.append({
    "Model": name,
    "CV_F1_macro": grid.best_score_,
    "Test_Accuracy": test_accuracy,
    "Test_F1_macro": test_f1_macro,
    "Test_Recall_class1": test_recall_class1,
    "Test_F1_class1": test_f1_class1
})

```

Best Params: {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 5, 'n_estimators': 200, 'subsample': 0.8}

	precision	recall	f1-score	support
0	0.97	0.90	0.93	1458
1	0.18	0.45	0.26	75
accuracy			0.87	1533
macro avg	0.58	0.67	0.60	1533
weighted avg	0.93	0.87	0.90	1533



Final Model Performance Comparison (Sorted by F1-macro)

```
In [34]: final_results_df = pd.DataFrame(final_results).sort_values(by="Test_F1_macro", ascending=False)
final_results_df
```

```
Out[34]:
```

	Model	CV_F1_macro	Test_Accuracy	Test_F1_macro	Test_Recall_class1	Test_F1_class1
0	Random Forest	0.939032	0.930855	0.677076	0.453333	0.390805
3	XGBoost	0.575858	0.874103	0.595865	0.453333	0.260536
2	Gradient Boosting	0.518302	0.945858	0.539804	0.066667	0.107527
1	Logistic Regression (SMOTE)	0.543797	0.767123	0.534242	0.613333	0.204900

```
In [35]: best_model_name = final_results_df.iloc[0]["Model"]
best_model_name
```

```
Out[35]: 'Random Forest'
```

```
In [36]: final_model = best_models[best_model_name]
```

Feature Selection Method 1: Chi-Square (Split → MinMax Scaling → SelectKBest → Feature Scores)

with Visualization of Chi-Square Feature Importance Scores

```
In [37]: #1) Ensure non-negative values for chi2
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 2) Chi-Square Feature Selection
k = 10
selector = SelectKBest(score_func=chi2, k=k)

selector.fit(X_train_scaled, y_train)

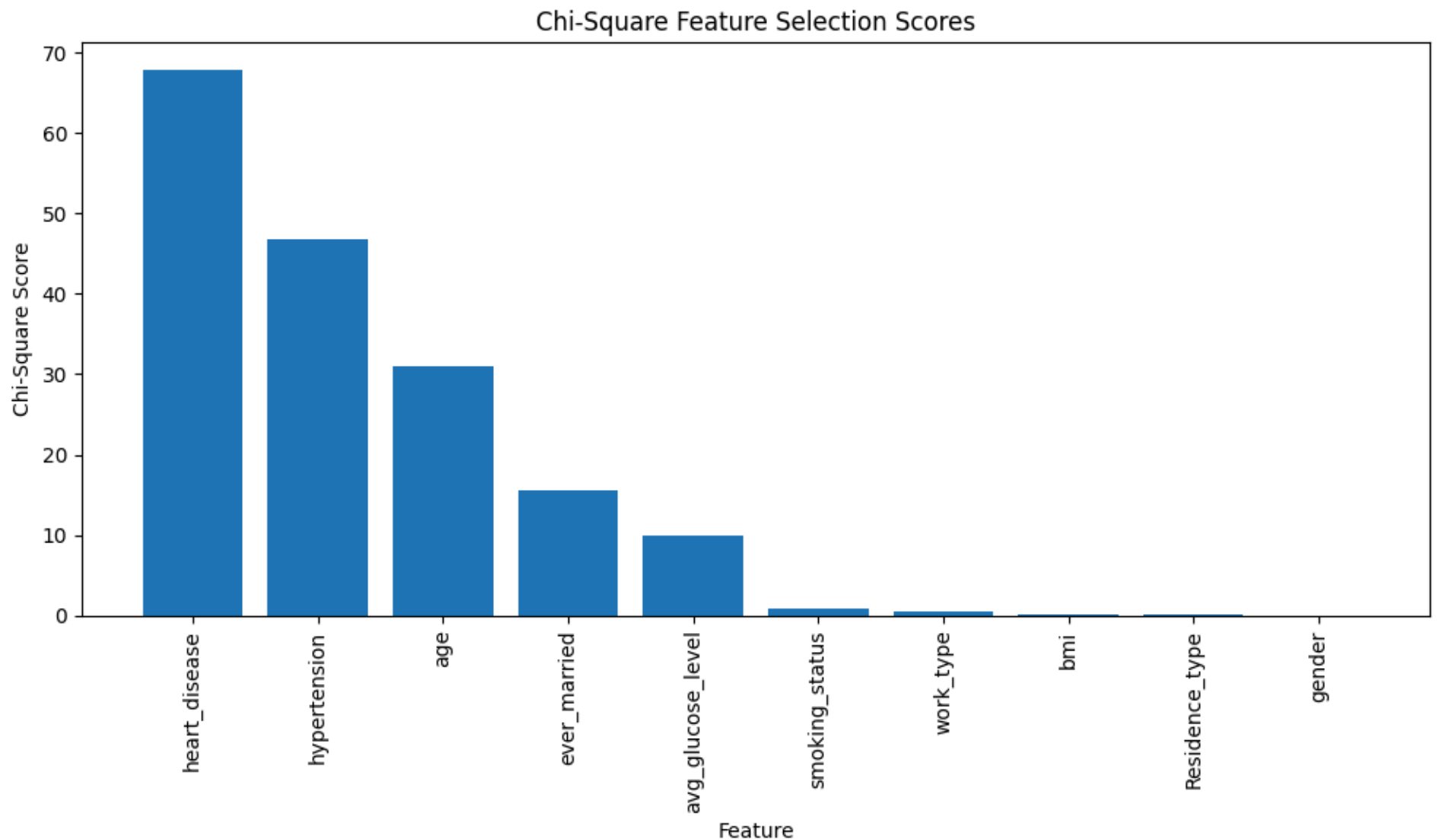
# 3) Scores + sorted table
data = pd.DataFrame({
    'Feature': X.columns,
    'Score': selector.scores_
}).sort_values(by='Score', ascending=False)

print("Top features (Chi-Square):", list(data.head(k)["Feature"]))

# 4) Plot bar chart
plt.figure(figsize=(10, 6))
plt.bar(data['Feature'], data['Score'])
plt.xlabel("Feature")
plt.ylabel("Chi-Square Score")
plt.title("Chi-Square Feature Selection Scores")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

# 5) Transform train/test (for later modeling)
X_train_chi = selector.transform(X_train_scaled)
X_test_chi = selector.transform(X_test_scaled)
```

Top features (Chi-Square): ['heart_disease', 'hypertension', 'age', 'ever_married', 'avg_glucose_level', 'smoking_status', 'work_type', 'bmi', 'Residence_type', 'gender']



Feature Selection Method 2: RFE Using Logistic Regression (Training Data Only)

```
In [38]: # Method 2: Recursive Feature Elimination (RFE)
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)

rfe_selector = RFE(
    estimator=LogisticRegression(max_iter=1000),
```

```

    n_features_to_select=5
)
rfe_selector.fit(X_train_scaled, y_train)
rfe_ranking = pd.DataFrame({
    "Feature": X.columns,
    "Rank": rfe_selector.ranking_
}).sort_values("Rank")

rfe_ranking

```

Out[38]:

	Feature	Rank
1	age	1
2	hypertension	1
3	heart_disease	1
5	work_type	1
7	avg_glucose_level	1
4	ever_married	2
9	smoking_status	3
8	bmi	4
0	gender	5
6	Residence_type	6

Visualization of Feature Importance Using RFE (Training Data Only)

```

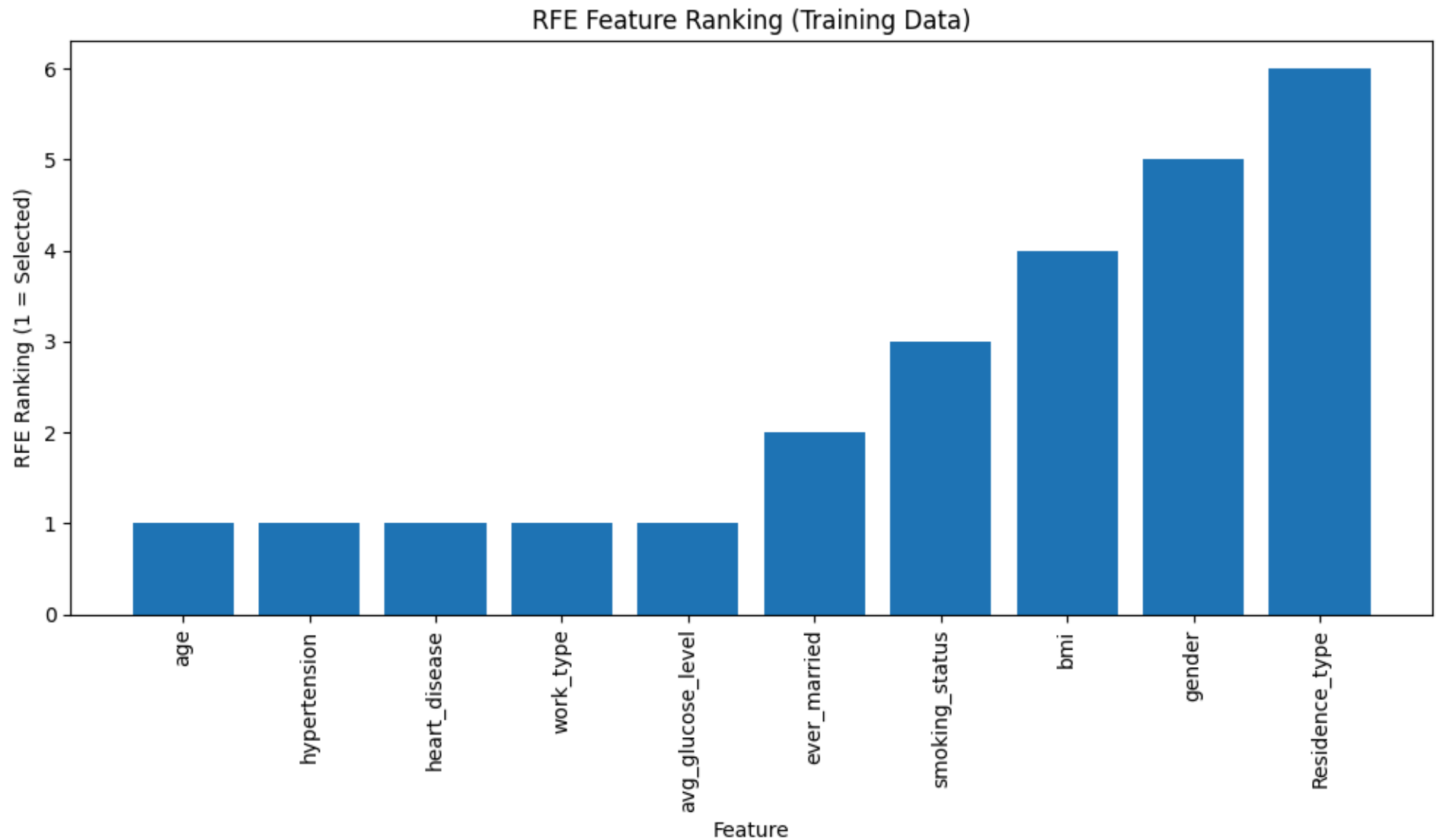
In [39]: # RFE Feature Ranking (based on training data only)

rfe_ranking = pd.DataFrame({
    "Feature": X.columns,
    "Ranking": rfe_selector.ranking_
}).sort_values(by="Ranking")

plt.figure(figsize=(10, 6))
plt.bar(rfe_ranking["Feature"], rfe_ranking["Ranking"])
plt.xlabel("Feature")
plt.ylabel("RFE Ranking (1 = Selected)")
plt.title("RFE Feature Ranking (Training Data)")

```

```
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

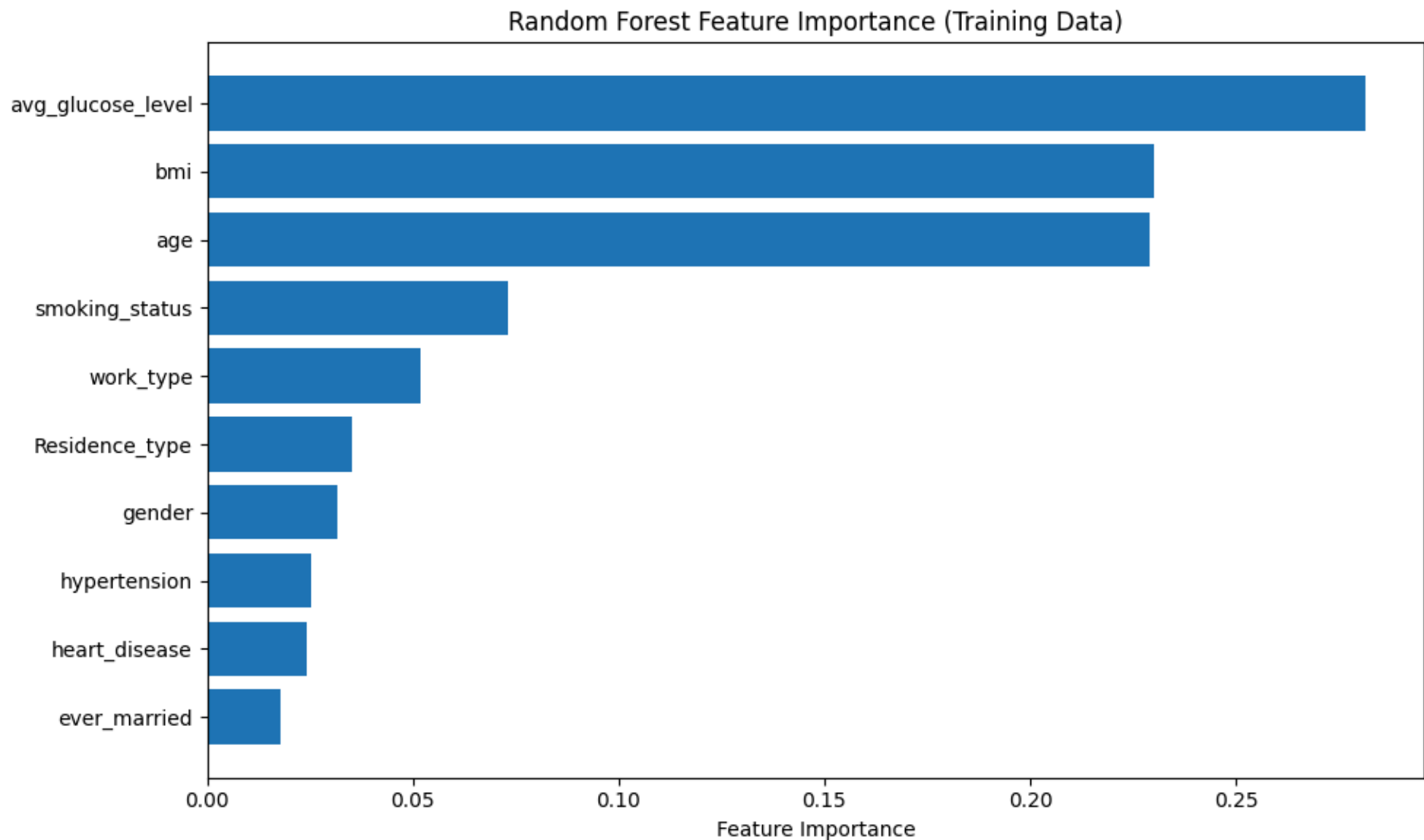


Model-Based Feature Importance: Random Forest (Training Data Only)

```
In [40]: # Random Forest Feature Importance (TRAIN only)

rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)  # تدريب على التدريب فقط
```

```
importances = rf.feature_importances_  
  
feature_names = X.columns if hasattr(X, "columns") else [f"Feature {i}" for i in range(X.shape[1])]   
  
df_importance = pd.DataFrame({  
    "Feature": feature_names,  
    "Importance": importances  
}).sort_values(by="Importance", ascending=False)  
  
plt.figure(figsize=(10, 6))  
plt.barh(df_importance["Feature"][::-1], df_importance["Importance"][::-1])  
plt.xlabel("Feature Importance")  
plt.title("Random Forest Feature Importance (Training Data)")  
plt.tight_layout()  
plt.show()
```



Model Evaluation Using RFE-Selected Features (Train-Only Feature Selection)

```
In [41]: # =====  
# Apply ML models on RFE-selected features + Evaluate (NO LEAKAGE)  
# RFE is trained on TRAIN only (with scaling), then applied to TEST  
# =====  
  
from sklearn.preprocessing import StandardScaler  
from sklearn.feature_selection import RFE  
from sklearn.linear_model import LogisticRegression
```

```

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import (
    classification_report, confusion_matrix,
    accuracy_score, precision_score, recall_score, f1_score
)
import pandas as pd

# =====
# 1) Scale TRAIN/TEST using the same scaler
# =====
scaler_rfe = StandardScaler()
X_train_scaled = scaler_rfe.fit_transform(X_train) # fit on TRAIN only
X_test_scaled = scaler_rfe.transform(X_test)      # transform TEST only

# =====
# 2) Fit RFE on TRAIN only
# =====
rfe_selector = RFE(
    estimator=LogisticRegression(max_iter=1000),
    n_features_to_select=5
)
rfe_selector.fit(X_train_scaled, y_train)

# Selected feature names (for reporting)
rfe_features = X.columns[rfe_selector.get_support()]
print("Top 5 Features (RFE):", list(rfe_features))

# Transform TRAIN/TEST into selected features
X_train_rfe = rfe_selector.transform(X_train_scaled)
X_test_rfe = rfe_selector.transform(X_test_scaled)

print("Original shape:", X_train.shape, "-> After RFE:", X_train_rfe.shape)

# =====
# 3) Define models (train on RFE features)
# =====
models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42, class_weight="balanced"),
    "Random Forest": RandomForestClassifier(n_estimators=200, random_state=42, class_weight="balanced"),
    "Logistic Regression": LogisticRegression(max_iter=2000, class_weight="balanced", random_state=42),
    "SVM (RBF)": SVC(kernel="rbf", class_weight="balanced", random_state=42)
}

results = []

```

```

# =====
# 4) Train + Evaluate each model
# =====
for name, model in models.items():
    model.fit(X_train_rfe, y_train)
    y_pred = model.predict(X_test_rfe)

    print("\n" + "="*45)
    print(f"Results using RFE features: {name}")
    print("="*45)
    print(classification_report(y_test, y_pred, zero_division=0))

    cm = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix:\n", cm)

    results.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision_weighted": precision_score(y_test, y_pred, average="weighted", zero_division=0),
        "Recall_weighted": recall_score(y_test, y_pred, average="weighted", zero_division=0),
        "F1_weighted": f1_score(y_test, y_pred, average="weighted", zero_division=0),
        "F1_macro": f1_score(y_test, y_pred, average="macro", zero_division=0)
    })

# =====
# 5) Final comparison table (sorted by F1_macro)
# =====
results_df = pd.DataFrame(results).sort_values(by="F1_macro", ascending=False)

print("\n==== Final Comparison (Models trained on RFE selected features) =====")
results_df

```

Top 5 Features (RFE): ['age', 'hypertension', 'heart_disease', 'work_type', 'avg_glucose_level']

Original shape: (3577, 10) -> After RFE: (3577, 5)

=====

Results using RFE features: Decision Tree

=====

	precision	recall	f1-score	support
0	0.95	0.95	0.95	1458
1	0.11	0.11	0.11	75
accuracy			0.91	1533
macro avg	0.53	0.53	0.53	1533
weighted avg	0.91	0.91	0.91	1533

Confusion Matrix:

```
[[1390  68]
 [  67   8]]
```

=====

Results using RFE features: Random Forest

=====

	precision	recall	f1-score	support
0	0.95	0.99	0.97	1458
1	0.11	0.03	0.04	75
accuracy			0.94	1533
macro avg	0.53	0.51	0.51	1533
weighted avg	0.91	0.94	0.92	1533

Confusion Matrix:

```
[[1442  16]
 [  73   2]]
```

=====

Results using RFE features: Logistic Regression

=====

	precision	recall	f1-score	support
0	0.98	0.73	0.84	1458
1	0.13	0.77	0.22	75
accuracy			0.73	1533
macro avg	0.56	0.75	0.53	1533
weighted avg	0.94	0.73	0.81	1533

Confusion Matrix:

```
[[1059 399]
 [ 17  58]]
```

Results using RFE features: SVM (RBF)

```
=====
              precision    recall  f1-score   support

     0           0.99       0.67       0.80       1458
     1           0.11       0.83       0.20         75

 accuracy          0.67       0.67       0.67       1533
 macro avg          0.55       0.75       0.50       1533
weighted avg          0.94       0.67       0.77       1533
=====
```

Confusion Matrix:

```
[[972 486]
 [ 13  62]]
```

==== Final Comparison (Models trained on RFE selected features) =====

Out[41]:

	Model	Accuracy	Precision_weighted	Recall_weighted	F1_weighted	F1_macro
0	Decision Tree	0.911937	0.912491	0.911937	0.912214	0.529824
2	Logistic Regression	0.728637	0.942259	0.728637	0.805608	0.526939
1	Random Forest	0.941944	0.910685	0.941944	0.924709	0.506537
3	SVM (RBF)	0.674494	0.944059	0.674494	0.766550	0.497390

Comparing Imbalance Handling Methods (ROS, RUS, SMOTE, SMOTE-Tomek) Using the Same Test Set

In [42]:

```
# =====
# Requirement: Handle class imbalance (Sampling methods) + Evaluate models
# Train on resampled TRAIN only, evaluate on original TEST only (No Leakage)
# =====

import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    classification_report,
```

```

    accuracy_score,
    precision_score,
    recall_score,
    f1_score
)

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTETomek

# -----
# 1) Define X, y
# -----
X = df.drop(columns=["stroke"])
y = df["stroke"]

# -----
# 2) Split first (IMPORTANT)
# -----
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.30,
    random_state=42,
    stratify=y
)

# -----
# 3) Define models
# -----
models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42, class_weight="balanced"),
    "Random Forest": RandomForestClassifier(n_estimators=200, random_state=42, class_weight="balanced"),
    "Logistic Regression": LogisticRegression(max_iter=2000, class_weight="balanced", random_state=42),
    "SVM (RBF)": SVC(kernel="rbf", class_weight="balanced", random_state=42)
}

# -----
# 4) Evaluation Function (Binary/Multi-class safe)
# -----
def evaluate_models(models, Xt, yt, X_eval, y_eval, method_name):
    results = []

```

```

for model_name, model in models.items():
    model.fit(Xt, yt)
    preds = model.predict(X_eval)

    acc = accuracy_score(y_eval, preds)
    prec = precision_score(y_eval, preds, average="weighted", zero_division=0)
    rec = recall_score(y_eval, preds, average="weighted", zero_division=0)
    f1 = f1_score(y_eval, preds, average="weighted", zero_division=0)

    results.append({
        "Sampling Method": method_name,
        "Model": model_name,
        "Accuracy": acc,
        "Precision (weighted)": prec,
        "Recall (weighted)": rec,
        "F1-score (weighted)": f1
    })

return results

# -----
# 5) Run experiments
# -----
all_results = []

# (A) Original (no resampling)
all_results += evaluate_models(models, X_train, y_train, X_test, y_test, "Original (Imbalanced)")

# (B) Random Oversampling
ros = RandomOverSampler(random_state=42)
X_ros, y_ros = ros.fit_resample(X_train, y_train)
all_results += evaluate_models(models, X_ros, y_ros, X_test, y_test, "Random Oversampling")

# (C) Random Undersampling
rus = RandomUnderSampler(random_state=42)
X_rus, y_rus = rus.fit_resample(X_train, y_train)
all_results += evaluate_models(models, X_rus, y_rus, X_test, y_test, "Random Undersampling")

# (D) SMOTE
smote = SMOTE(random_state=42)
X_sm, y_sm = smote.fit_resample(X_train, y_train)
all_results += evaluate_models(models, X_sm, y_sm, X_test, y_test, "SMOTE")

# (E) SMOTE + Tomek Links
smt = SMOTETomek(random_state=42)

```

```

X_smt, y_smt = smt.fit_resample(X_train, y_train)
all_results += evaluate_models(models, X_smt, y_smt, X_test, y_test, "SMOTE + Tomek Links")

# -----
# 6) Final comparison table
# -----
results_df = pd.DataFrame(all_results)

summary = results_df.pivot_table(
    index="Sampling Method",
    columns="Model",
    values=["Accuracy", "Precision (weighted)", "Recall (weighted)", "F1-score (weighted)"]
).round(4)

print("==== Final Comparison (Sampling Methods vs Models) =====")
summary

```

==== Final Comparison (Sampling Methods vs Models) =====

Out[42]:

Model	Accuracy				F1-score (weighted)				Precision (weighted)				
	Decision Tree	Logistic Regression	Random Forest	SVM (RBF)	Decision Tree	Logistic Regression	Random Forest	SVM (RBF)	Decision Tree	Logistic Regression	Random Forest	SVM (RBF)	Decision Tree
Sampling Method													
Original (Imbalanced)	0.9217	0.7273	0.9504	0.6869	0.9185	0.8047	0.9269	0.7758	0.9154	0.9422	0.9045	0.9415	0.9217
Random Oversampling	0.9178	0.7306	0.9432	0.6908	0.9150	0.8070	0.9264	0.7786	0.9123	0.9414	0.9140	0.9407	0.9178
Random Undersampling	0.6999	0.7065	0.6882	0.6804	0.7852	0.7899	0.7767	0.7709	0.9334	0.9423	0.9436	0.9433	0.6999
SMOTE	0.8689	0.7678	0.8858	0.6875	0.8906	0.8318	0.8981	0.7762	0.9161	0.9333	0.9114	0.9435	0.8689
SMOTE + Tomek Links	0.8611	0.7678	0.8832	0.6869	0.8859	0.8318	0.8970	0.7757	0.9154	0.9333	0.9121	0.9435	0.8611

Conclusion

In this project, multiple machine learning models were developed and evaluated to predict stroke occurrence using a healthcare dataset characterized by class imbalance. To address this challenge, several imbalance handling techniques were applied, including Random Oversampling, Random

Undersampling, SMOTE, and SMOTE–Tomek, all of which were carefully applied to the training data only to avoid data leakage.

GridSearchCV was used to optimize model hyperparameters, and model performance was evaluated on a fixed test set to ensure a fair comparison. Due to the imbalanced nature of the dataset, F1-macro was selected as the primary evaluation metric, as it provides a balanced assessment across both majority and minority classes.

Additionally, two feature selection methods were employed: Chi-Square and Recursive Feature Elimination (RFE). The results demonstrated that feature selection can reduce dimensionality while maintaining competitive model performance, particularly when combined with appropriate scaling and training-only feature selection.

Overall, the experiments highlight the importance of handling class imbalance, selecting appropriate evaluation metrics, and applying feature selection in a methodologically sound manner. The final results indicate that models trained with balanced data or class-weighted approaches achieved more reliable and robust performance for stroke prediction.