In [116…
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, Confusi
from xgboost import XGBClassifier
from sklearn.metrics import precision_score, recall_score
from sklearn.feature_selection import mutual_info_classif
```

In [8]:
```python
df = pd.read_csv("/Users/rangli/Documents/DSC680/Project3/brain_stroke.csv")
df.head()
```

Out[8]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_ty
0	Male	67.0	0	1	Yes	Private	Urb
1	Male	80.0	0	1	Yes	Private	Ru
2	Female	49.0	0	0	Yes	Private	Urb
3	Female	79.0	1	0	Yes	Self-employed	Ru
4	Male	81.0	0	0	Yes	Private	Urb

In [ ]:
```python
# check dataset info
```

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

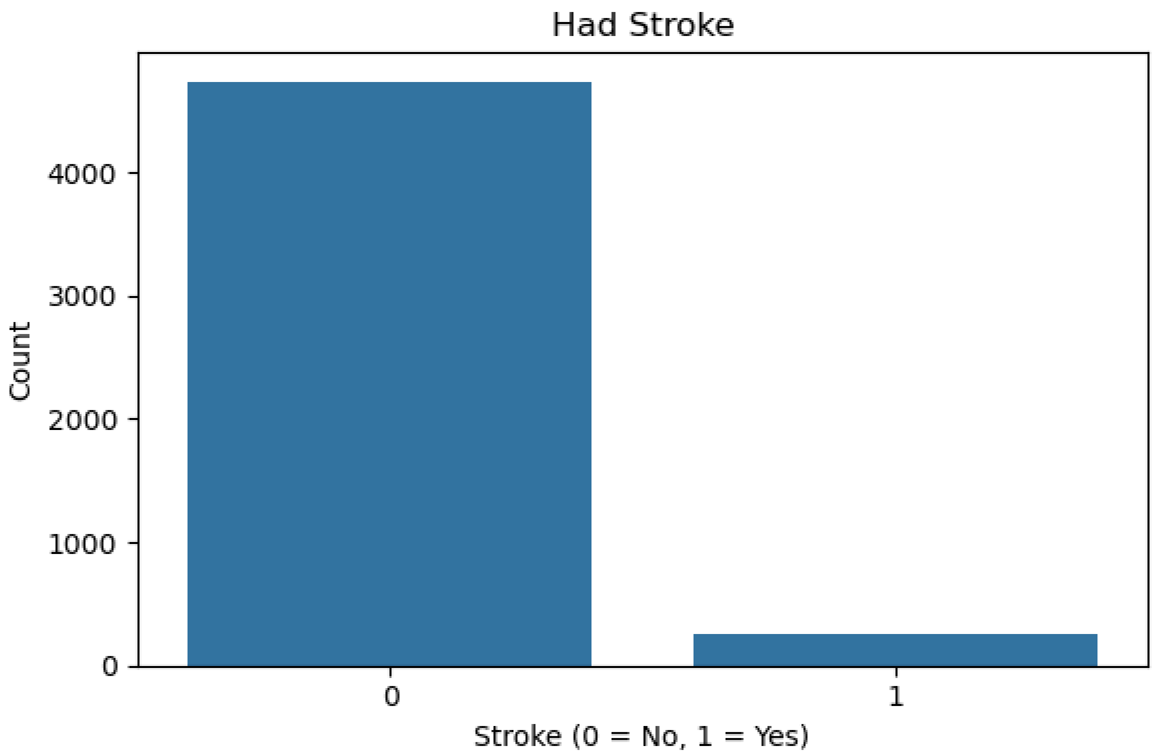
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4981 entries, 0 to 4980
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   gender             4981 non-null   object 
 1   age                4981 non-null   float64
 2   hypertension       4981 non-null   int64  
 3   heart_disease      4981 non-null   int64  
 4   ever_married       4981 non-null   object 
 5   work_type          4981 non-null   object 
 6   Residence_type     4981 non-null   object 
 7   avg_glucose_level  4981 non-null   float64
 8   bmi                4981 non-null   float64
 9   smoking_status     4981 non-null   object 
 10  stroke             4981 non-null   int64  
dtypes: float64(3), int64(3), object(5)
memory usage: 428.2+ KB
```

In [15]:
```python
df.duplicated().sum()
```

Out[15]: 0

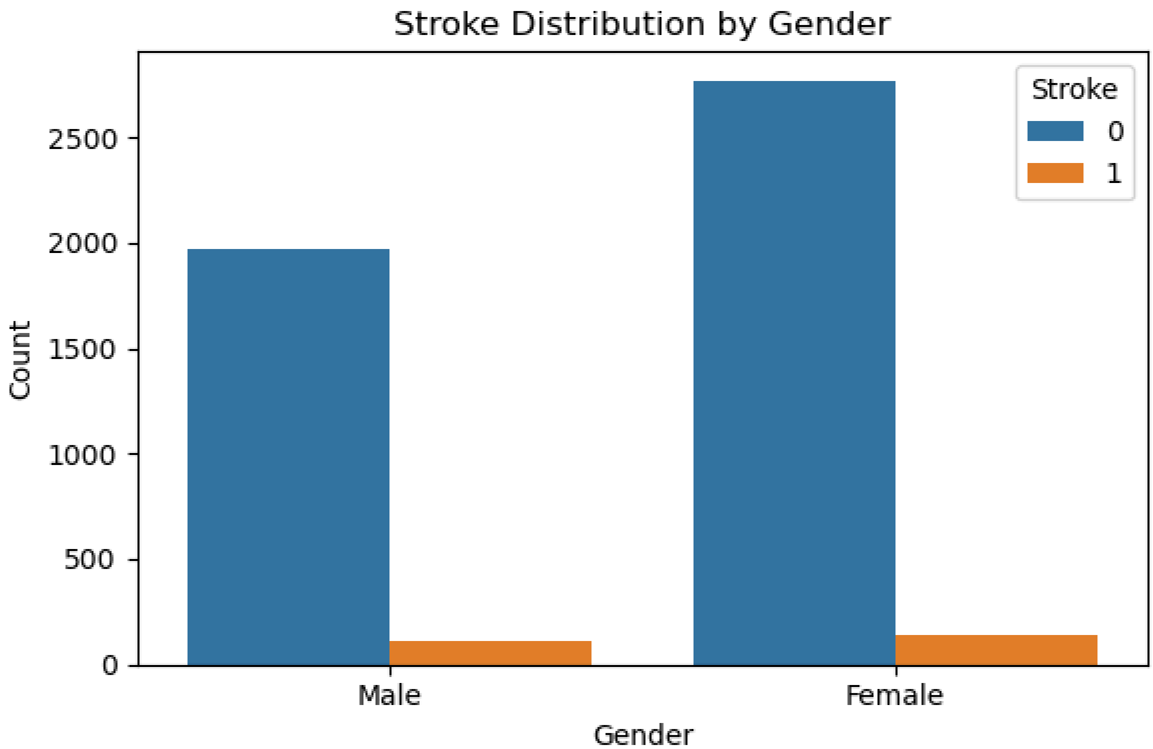
In [ ]:
```
# check distributions of stroke
```

In [21]:
```
plt.figure(figsize=(6, 4))
sns.countplot(x='stroke', data=df)
plt.title('Had Stroke')
plt.xlabel('Stroke (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```



In [ ]:
```
# gender distribution
```

In [30]:
```
plt.figure(figsize=(6, 4))
sns.countplot(x='gender', hue='stroke', data=df)
plt.title('Stroke Distribution by Gender')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.legend(title='Stroke')
plt.tight_layout()
plt.show()
```



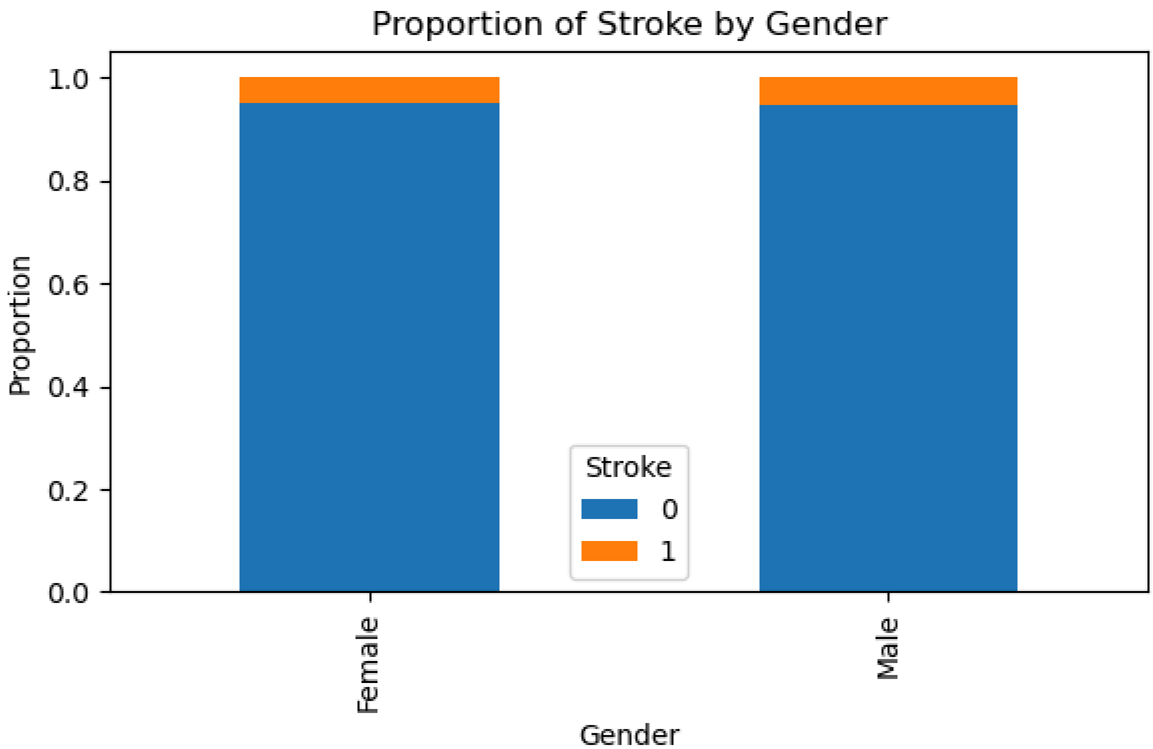
In [35]:
```
# calculate proportions of stroke vs gender
```

In [36]:
```
gender_stroke = df.groupby('gender')['stroke'].value_counts(normalize=True).
gender_stroke
```

Out[36]:

stroke	0	1
gender		
Female	0.951840	0.048160
Male	0.947927	0.052073

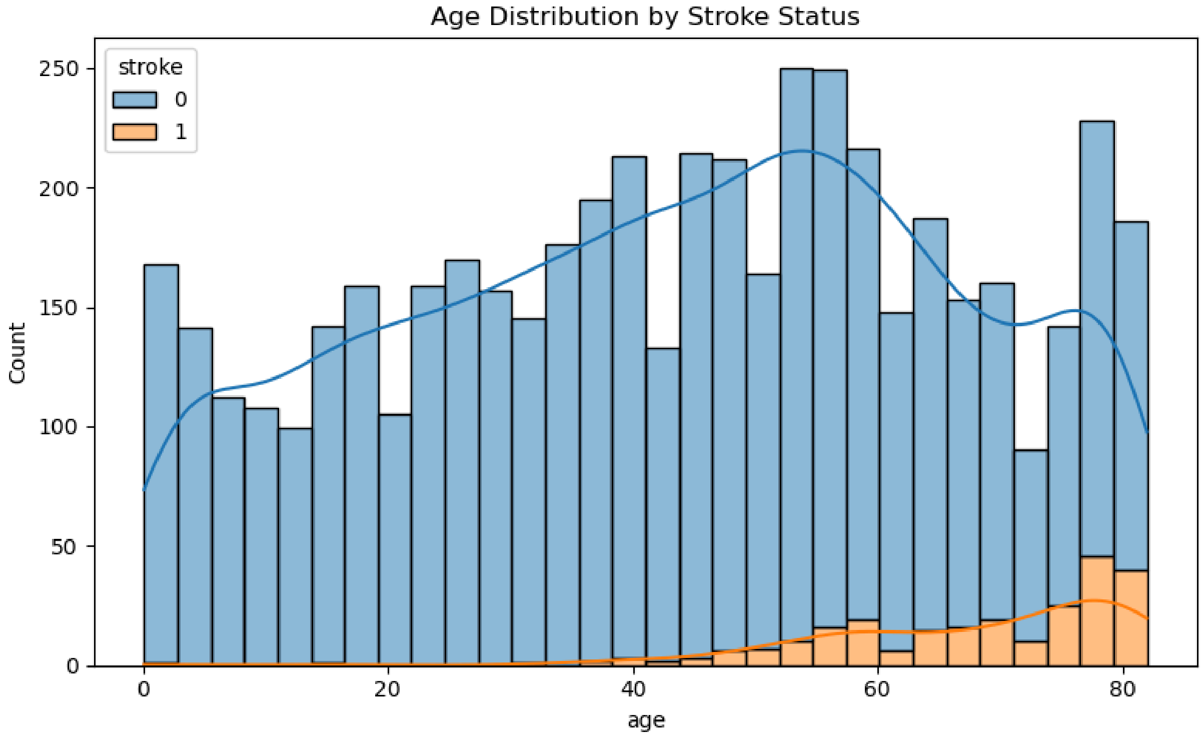
In [37]:
```
gender_stroke.plot(kind='bar', stacked=True, figsize=(6, 4))
plt.title('Proportion of Stroke by Gender')
plt.xlabel('Gender')
plt.ylabel('Proportion')
plt.legend(title='Stroke')
plt.tight_layout()
plt.show()
```



According to the caculation of the proportions of strokes groups by gender. Stroke is rare in both genders, the plot reflecting about 5% overall stroke rate in each group, with the majority of both groups did not experience stroke. However the proportion of stroke in mens is slightly higher than womens, but with no significant difference.

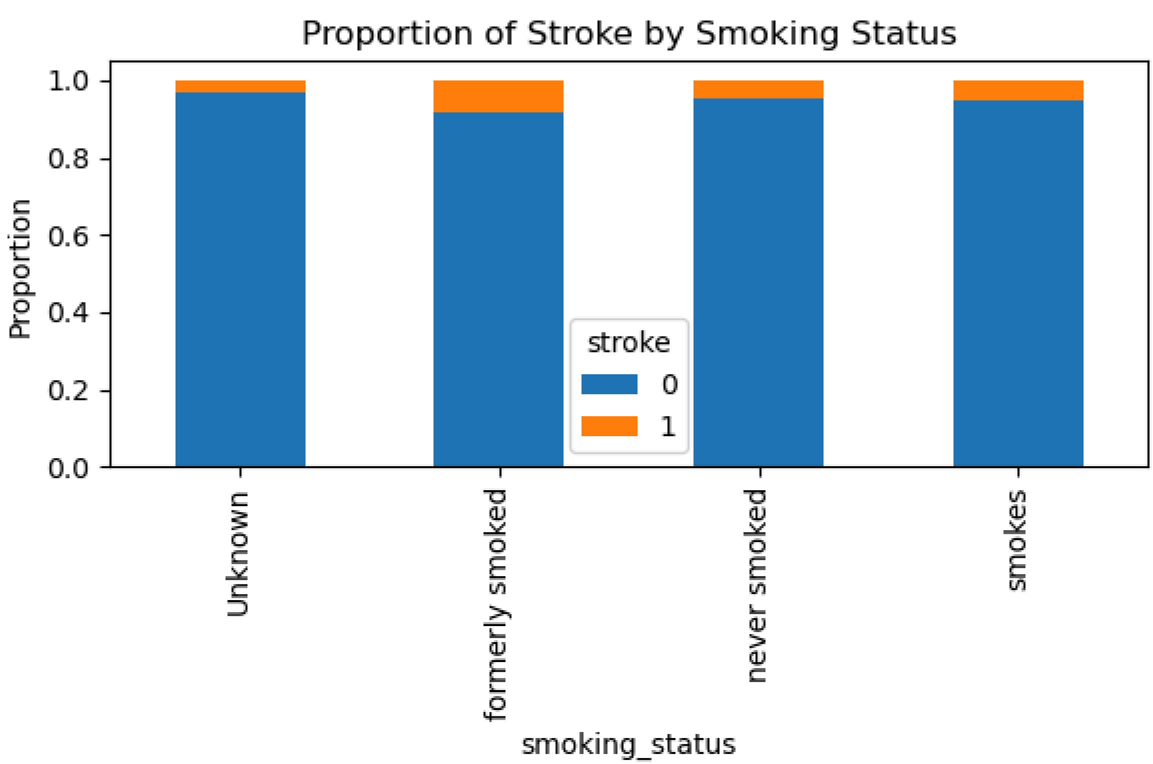
In [38]:
```
# age distribution by stroke status
```

In [23]:
```
plt.figure(figsize=(8, 5))
sns.histplot(data=df, x='age', hue='stroke', bins=30, kde=True, multiple='st
plt.title('Age Distribution by Stroke Status')
plt.tight_layout()
plt.show()
```



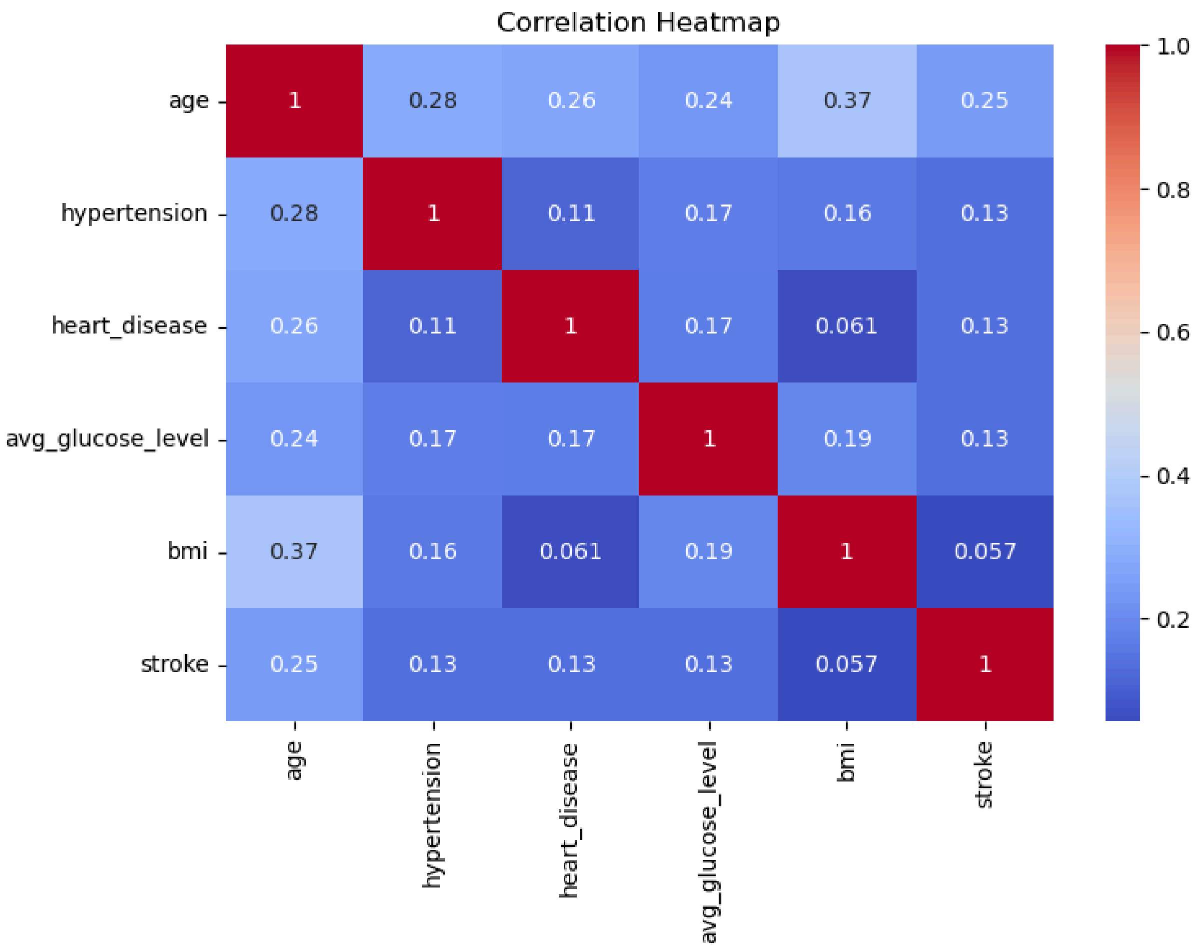
In [ ]:
```
# proportions of stroke by smoking
```

In [40]:
```
prop = df.groupby('smoking_status')['stroke'].value_counts(normalize=True).u
prop.plot(kind='bar', stacked=True, figsize=(6, 4))
plt.title('Proportion of Stroke by Smoking Status')
plt.ylabel('Proportion')
plt.tight_layout()
plt.show()
```



In [41]:
```python
# heatmap for correlation
```

In [42]:
```python
plt.figure(figsize=(8, 6))
numerical_cols = ['age', 'hypertension', 'heart_disease', 'avg_glucose_level
sns.heatmap(df[numerical_cols].corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.tight_layout()
plt.show()
```



In [43]:
```python
# encoding categorical variable
```

In [45]:
```python
categorical_cols = ['gender', 'ever_married', 'work_type', 'Residence_type',
```

In [46]:
```python
df_encoded = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
```

In [48]:
```python
df_encoded.head()
```

Out[48]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender_Male
0	67.0	0	1	228.69	36.6	1	True
1	80.0	0	1	105.92	32.5	1	True
2	49.0	0	0	171.23	34.4	1	False
3	79.0	1	0	174.12	24.0	1	False
4	81.0	0	0	186.21	29.0	1	True

In [53]:
```
# define features
```
In [50]:
```
X = df_encoded.drop('stroke', axis=1)
y = df_encoded['stroke']
```
In [55]:
```
# split into train and test datas
```
In [51]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```
In [54]:
```
# apply SMOTE on the training data to resample
```
In [52]:
```
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```
In [ ]:
```
# Check class distribution before and after SMOTE
```
In [56]:
```
before_resample = y_train.value_counts(normalize=True)
after_resample = y_train_resampled.value_counts(normalize=True)
```
In [57]:
```
before_resample
```
Out[57]:
```
stroke
0    0.950301
1    0.049699
Name: proportion, dtype: float64
```
In [58]:
```
after_resample
```
Out[58]:
```
stroke
0    0.5
1    0.5
Name: proportion, dtype: float64
```
In [61]:
```
# train models on resampled data
```
In [62]:
```
model_random = RandomForestClassifier(random_state=42)
model_random.fit(X_train_resampled, y_train_resampled)
```

Out[62]:

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [63]:

```python
y_pred_random = model_random.predict(X_test)
```

In [65]:

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

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       947
           1       0.17      0.12      0.14        50

    accuracy                           0.93       997
   macro avg       0.56      0.54      0.55       997
weighted avg       0.91      0.93      0.92       997
```

In [66]:

```python
# train a random forrest model with class weighing to account for imbalance
```

In [67]:

```python
model_r_weigh = RandomForestClassifier(class_weight='balanced', random_state
model_r_weigh.fit(X_train, y_train)
```

Out[67]:

```
RandomForestClassifier
RandomForestClassifier(class_weight='balanced', random_state=42)
```

In [70]:

```python
y_pred_r_weigh = model_r_weigh.predict(X_test)
```

In [71]:

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

```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       947
           1       0.00      0.00      0.00        50

    accuracy                           0.95       997
   macro avg       0.47      0.50      0.49       997
weighted avg       0.90      0.95      0.92       997
```

In [72]:

```python
# train a logistic regression model with class weigh
```

```
The history saving thread hit an unexpected error (OperationalError('attempt
to write a readonly database')).History will not be written to the database.
```

In [75]:

```python
log_model = LogisticRegression(class_weight='balanced', max_iter=1000, rando
log_model.fit(X_train, y_train)
```

Out[75]:

```
▾ LogisticRegression
LogisticRegression(class_weight='balanced', max_iter=1000, random_s
tate=42)
```

In [76]:

```
y_pred_log = log_model.predict(X_test)
y_prob_log = log_model.predict_proba(X_test)[:, 1]
```

In [77]:

```
report = classification_report(y_test, y_pred_log, target_names=["No Stroke"
print(report)
```

```
              precision    recall  f1-score   support

   No Stroke       0.99      0.75      0.85       947
      Stroke       0.15      0.84      0.25        50

    accuracy                           0.75       997
   macro avg       0.57      0.79      0.55       997
weighted avg       0.95      0.75      0.82       997
```

In [78]:

```
# Calculate scale_pos_weight for imbalance handling
```

In [80]:

```
neg, pos = y_train.value_counts()
scale_pos_weight = neg / pos
scale_pos_weight
```

Out[80]: 19.12121212121212

In [81]:

```
# Train XGBoost model
```

In [87]:

```
xgb_model = XGBClassifier(scale_pos_weight=scale_pos_weight, use_label_encoc
xgb_model.fit(X_train, y_train)
```

```
/opt/anaconda3/lib/python3.12/site-packages/xgboost/training.py:183: UserWar
ning: [15:38:32] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:
738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

Out[87]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_ro
unds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max
_bin=None,
```

In [88]:

```
# Predict and evaluate
```

In [89]:

```
y_pred_xgb = xgb_model.predict(X_test)
y_prob_xgb = xgb_model.predict_proba(X_test)[:, 1]
```

In [90]:

```
# Generate classification report
```

In [94]:

```
xbg_report = classification_report(y_test, y_pred_xgb, target_names=["No Str
print(xbg_report)
```

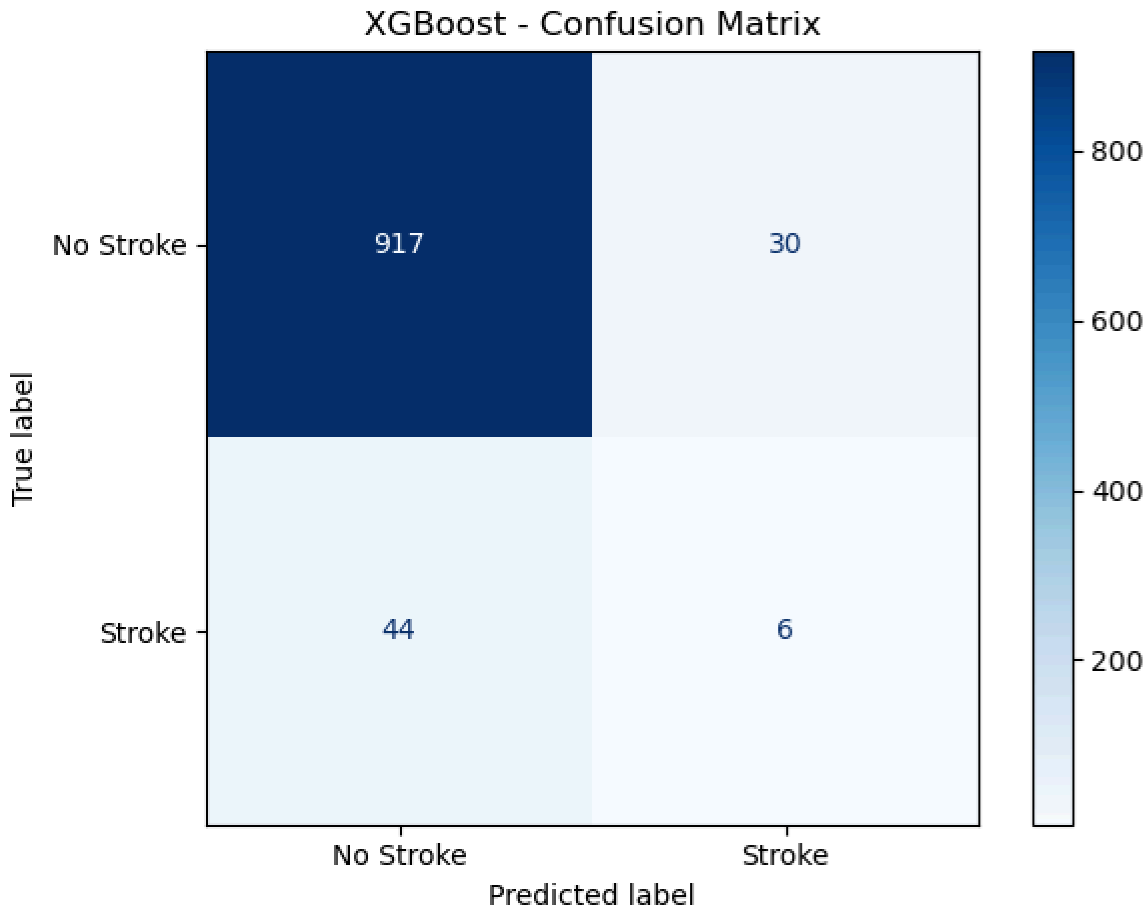
```
              precision    recall  f1-score   support

   No Stroke       0.95      0.97      0.96       947
      Stroke       0.17      0.12      0.14        50

    accuracy                           0.93       997
   macro avg       0.56      0.54      0.55       997
weighted avg       0.91      0.93      0.92       997
```

In [95]:

```
# confusion matrix
```

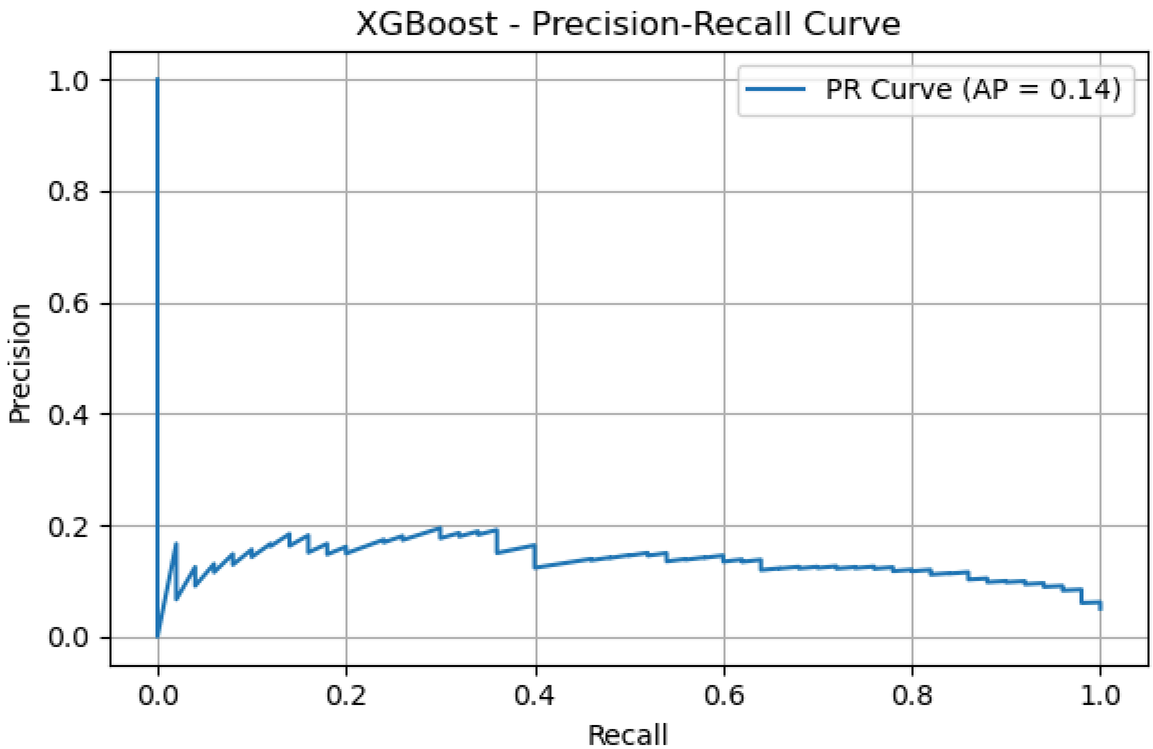
In [96]:

```
cm = confusion_matrix(y_test, y_pred_xgb)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["No Strok
disp.plot(cmap=plt.cm.Blues)
plt.title("XGBoost - Confusion Matrix")
plt.tight_layout()
plt.show()
```



In [97]:
```
# precision-recall curve
```

In [98]:
```
precision, recall, _ = precision_recall_curve(y_test, y_prob_xgb)
avg_precision = average_precision_score(y_test, y_prob_xgb)

plt.figure(figsize=(6, 4))
plt.plot(recall, precision, label=f"PR Curve (AP = {avg_precision:.2f})")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("XGBoost - Precision-Recall Curve")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [99]:
```
threshold = 0.3
y_pred_thresh = (y_prob_xgb >= threshold).astype(int)
```

In [100…
```
report_thresh = classification_report(y_test, y_pred_thresh, target_names=["
print(f"Classification report at threshold = {threshold}:\n")
print(report_thresh)
```

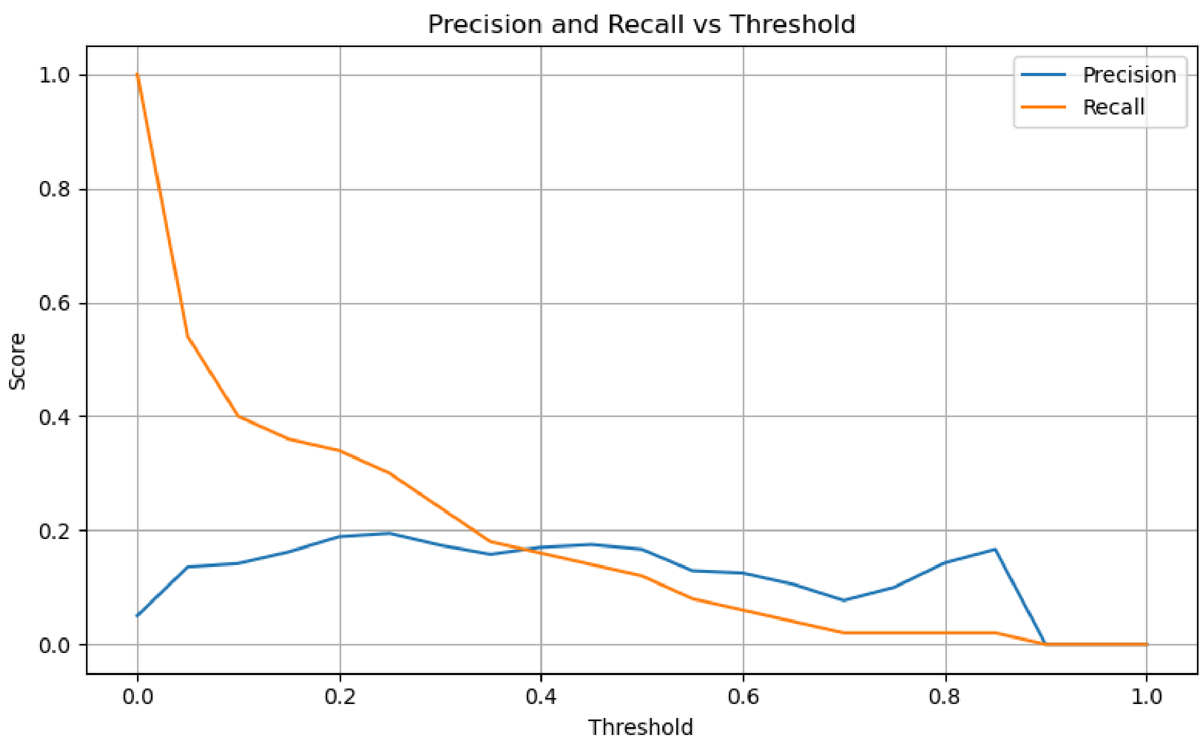
```
Classification report at threshold = 0.3:

              precision    recall  f1-score   support

   No Stroke       0.96      0.94      0.95       947
      Stroke       0.17      0.24      0.20        50

    accuracy                           0.90       997
   macro avg       0.57      0.59      0.58       997
weighted avg       0.92      0.90      0.91       997
```

In [101…
```
thresholds_range = np.arange(0.0, 1.01, 0.05)
```

In [105…
```
precisions = []
recalls = []
for t in thresholds_range:
    y_pred_t = (y_prob_xgb >= t).astype(int)
    precisions.append(precision_score(y_test, y_pred_t))
    recalls.append(recall_score(y_test, y_pred_t))
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/metrics/_classification.
py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to
0.0 due to no predicted samples. Use `zero_division` parameter to control th
is behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

In [106…
```python
plt.figure(figsize=(8, 5))
plt.plot(thresholds_range, precisions, label="Precision")
plt.plot(thresholds_range, recalls, label="Recall")
plt.xlabel("Threshold")
plt.ylabel("Score")
plt.title("Precision and Recall vs Threshold")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [107…
```python
# apply precision of 0.2 to improve accuracy
```

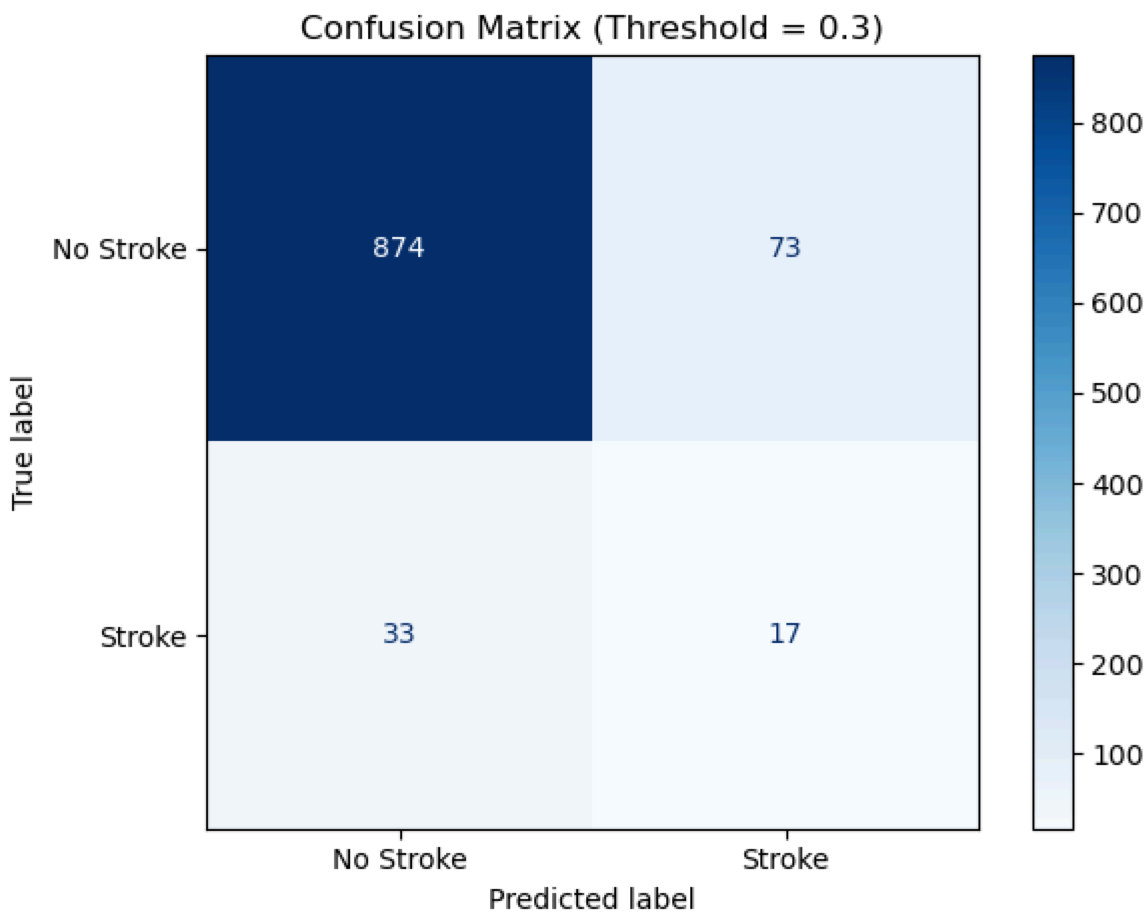
In [108…
```python
threshold_improve = 0.2
y_pred_adjusted = (y_prob_xgb >= threshold_improve).astype(int)
```

In [114…
```python
print(f"Classification report at threshold = {threshold_improve}:\n")
print(classification_report(y_test, y_pred_adjusted, target_names=["No Strok
```

```
Classification report at threshold = 0.2:

              precision    recall  f1-score   support

   No Stroke       0.96      0.92      0.94       947
      Stroke       0.19      0.34      0.24        50

    accuracy                           0.89       997
   macro avg       0.58      0.63      0.59       997
weighted avg       0.92      0.89      0.91       997
```

In [112…

```python
cm = confusion_matrix(y_test, y_pred_adjusted)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["No Strok
disp.plot(cmap=plt.cm.Blues)
plt.title(f"Confusion Matrix (Threshold = {threshold})")
plt.tight_layout()
plt.show()
```



The higher recall means that more cases are detected. The F1 score is improved too, showing better balance between catching strokes and avoiding false positives. Meanwhile precision slightly improves too — rare and desirable. However the small drop in accuracy is expected and acceptable — because we're prioritizing the minority class (stroke), which is more important in medical contexts.

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```