Stroke Risk Prediction (Kaggle Dataset)

Goal: Predict likelihood of stroke using demographic and medical indicators.

Key Findings

- Logistic Regression achieved ROC-AUC = 0.84, detecting 80% of positive cases.
- Random Forest performed better on overall accuracy (0.95) but missed minority cases.
- Strong predictors: age, glucose level, hypertension, heart disease, BMI.
- Illustrates the challenges of imbalanced medical datasets and the importance of recall in healthcare ML.

Tools: Python, Pandas, scikit-learn, Seaborn, Matplotlib
Techniques: EDA · Feature Encoding · Standardization · Baseline ML (LogReg & RF)

Dataset: Stroke Prediction Dataset - Kaggle

In [33]:
```
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 sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

# load dataset
df = pd.read_csv("healthcare-dataset-stroke-data.csv")

# inspect
df.head()
df.info()
df.describe(include="all").T
```

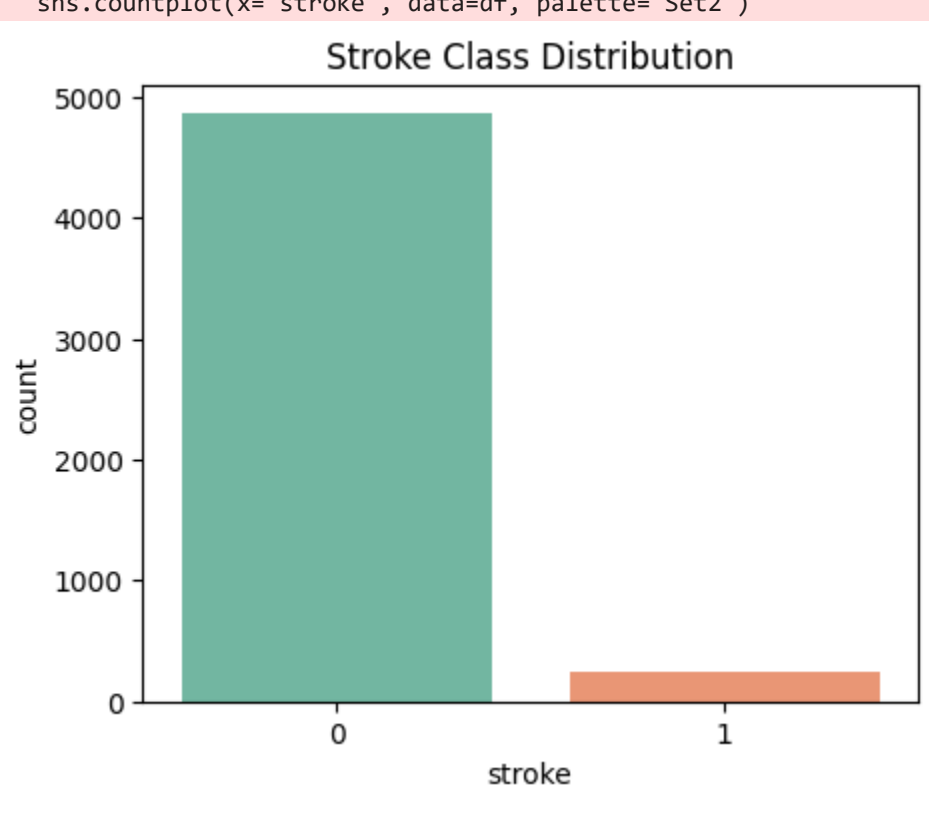
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5110 non-null   int64
 1   gender             5110 non-null   object
 2  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

Out[33]:

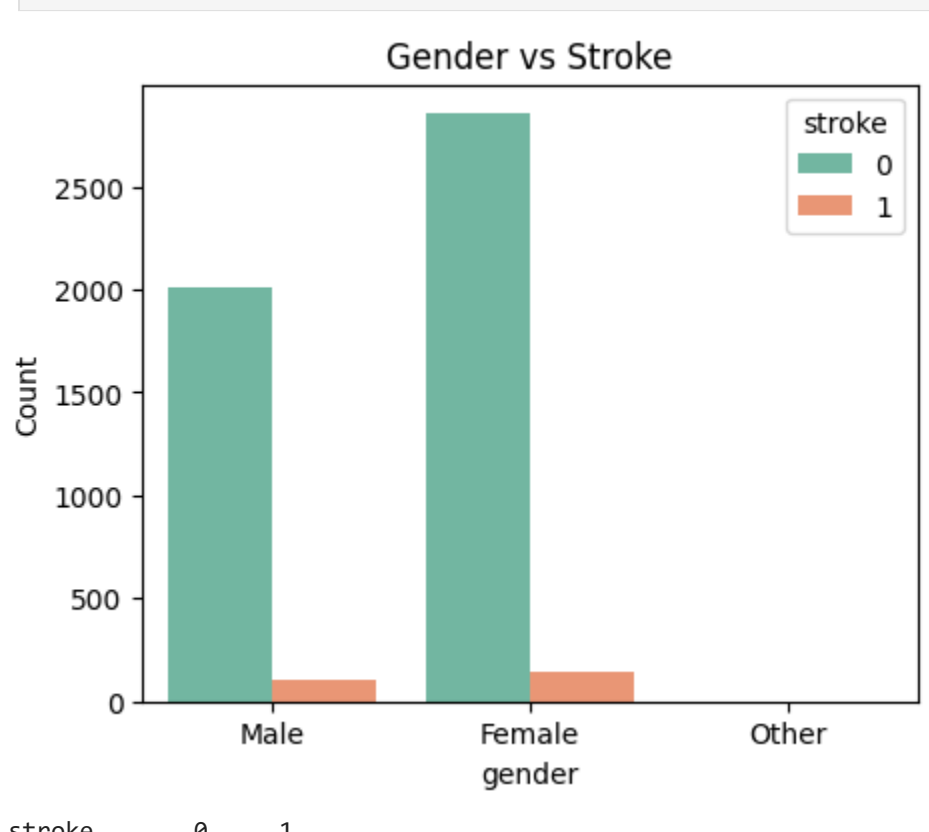
	count	unique	top	freq	mean	std	min	25%	50%	75%	max
id	5110.0	NaN	NaN	NaN	36517.829354	21161.721625	67.0	17741.25	36932.0	54682.0	72940.0
gender	5110	3	Female	2994	NaN	NaN	NaN	NaN	NaN	NaN	NaN
age	5110.0	NaN	NaN	NaN	43.226614	22.612647	0.08	25.0	45.0	61.0	82.0
hypertension	5110.0	NaN	NaN	NaN	0.097456	0.296607	0.0	0.0	0.0	0.0	1.0
heart_disease	5110.0	NaN	NaN	NaN	0.054012	0.226063	0.0	0.0	0.0	0.0	1.0
ever_married	5110	2	Yes	3353	NaN	NaN	NaN	NaN	NaN	NaN	NaN
work_type	5110	5	Private	2925	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Residence_type	5110	2	Urban	2596	NaN	NaN	NaN	NaN	NaN	NaN	NaN
avg_glucose_level	5110.0	NaN	NaN	NaN	106.147677	45.28356	55.12	77.245	91.885	114.09	271.74
bmi	4909.0	NaN	NaN	NaN	28.893237	7.854067	10.3	23.5	28.1	33.1	97.6
smoking_status	5110	4	never smoked	1892	NaN	NaN	NaN	NaN	NaN	NaN	NaN
stroke	5110.0	NaN	NaN	NaN	0.048728	0.21532	0.0	0.0	0.0	0.0	1.0

In [34]:
```
# EDA

# Checking for missing values
df.isna().sum()

# Class distribution
plt.figure(figsize=(5,4))
sns.countplot(x='stroke', data=df, palette='Set2')
plt.title('Stroke Class Distribution')
plt.show()
```

```
/var/folders/yr/_hbn72w179q5zw1nlsq86xxc0000gn/T/ipykernel_4516/3065397531.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(x='stroke', data=df, palette='Set2')
```

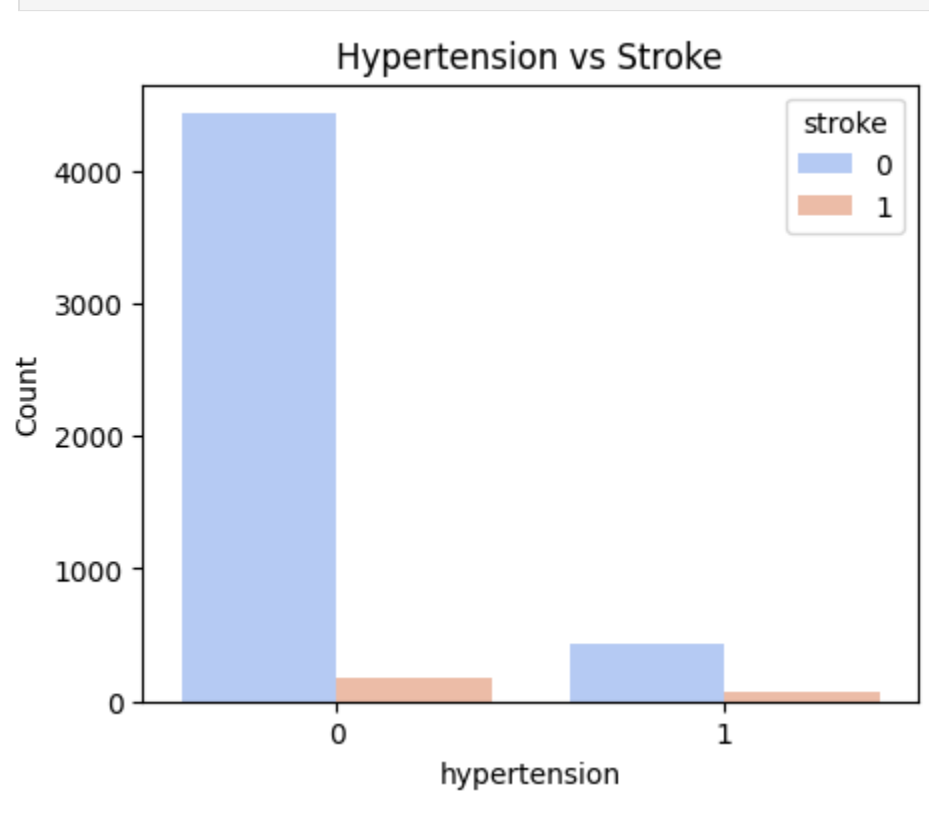


In [35]:
```
plt.figure(figsize=(5,4))
sns.countplot(x='gender', hue='stroke', data=df, palette='Set2')
plt.title('Gender vs Stroke')
plt.ylabel('Count')
plt.show()

# Calculate percentages
gender_stroke = pd.crosstab(df['gender'], df['stroke'], normalize='index') * 100
print(gender_stroke.round(2))
```




```
stroke       0     1
gender
Female   95.29  4.71
Male     94.89  5.11
Other   100.00  0.00
```

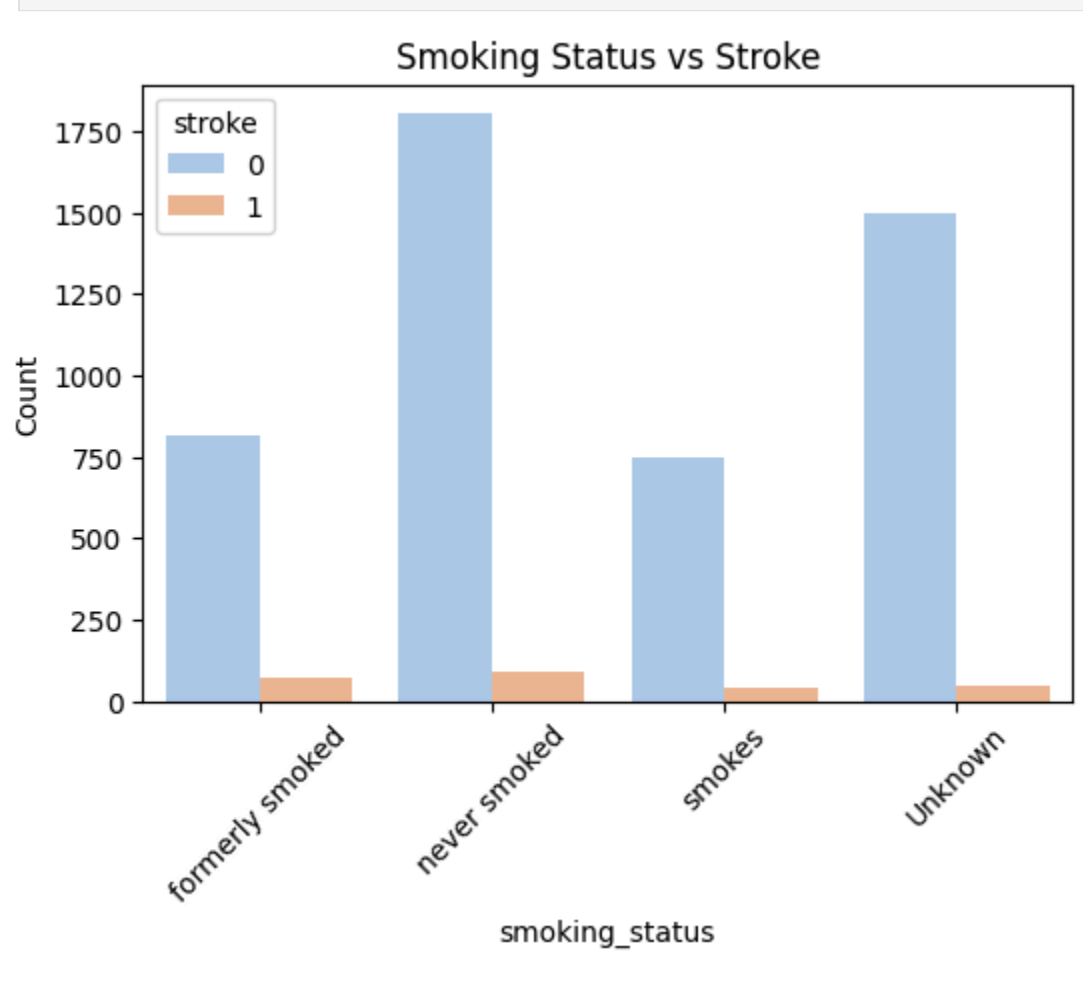
In [36]:
```
plt.figure(figsize=(5,4))
sns.countplot(x='hypertension', hue='stroke', data=df, palette='coolwarm')
plt.title('Hypertension vs Stroke')
plt.ylabel('Count')
plt.show()

pd.crosstab(df['hypertension'], df['stroke'], normalize='index') * 100
```



Out[36]:

stroke	0	1
hypertension		
0	96.032090	3.967910
1	86.746988	13.253012

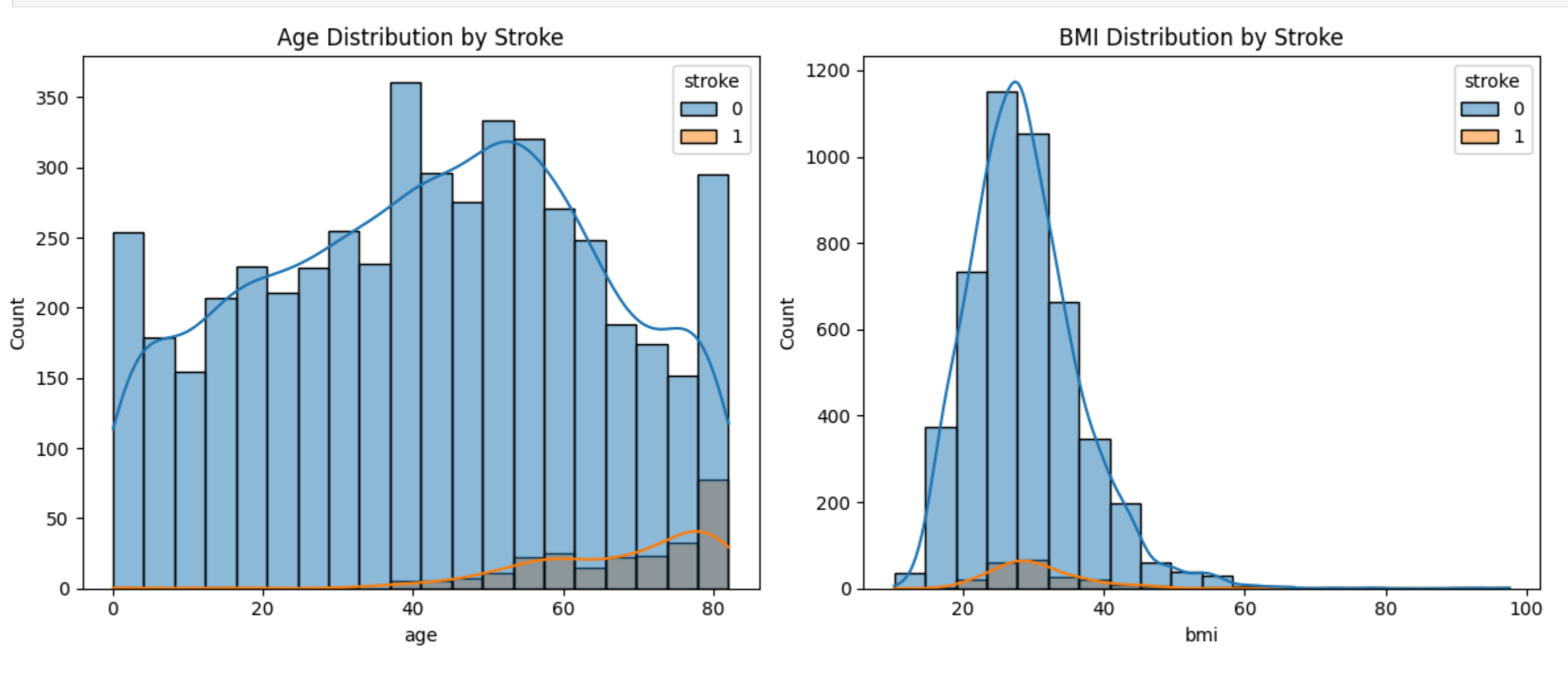
In [37]:
```
plt.figure(figsize=(6,4))
sns.countplot(x='smoking_status', hue='stroke', data=df, palette='pastel')
plt.title('Smoking Status vs Stroke')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()

pd.crosstab(df['smoking_status'], df['stroke'], normalize='index') * 100
```



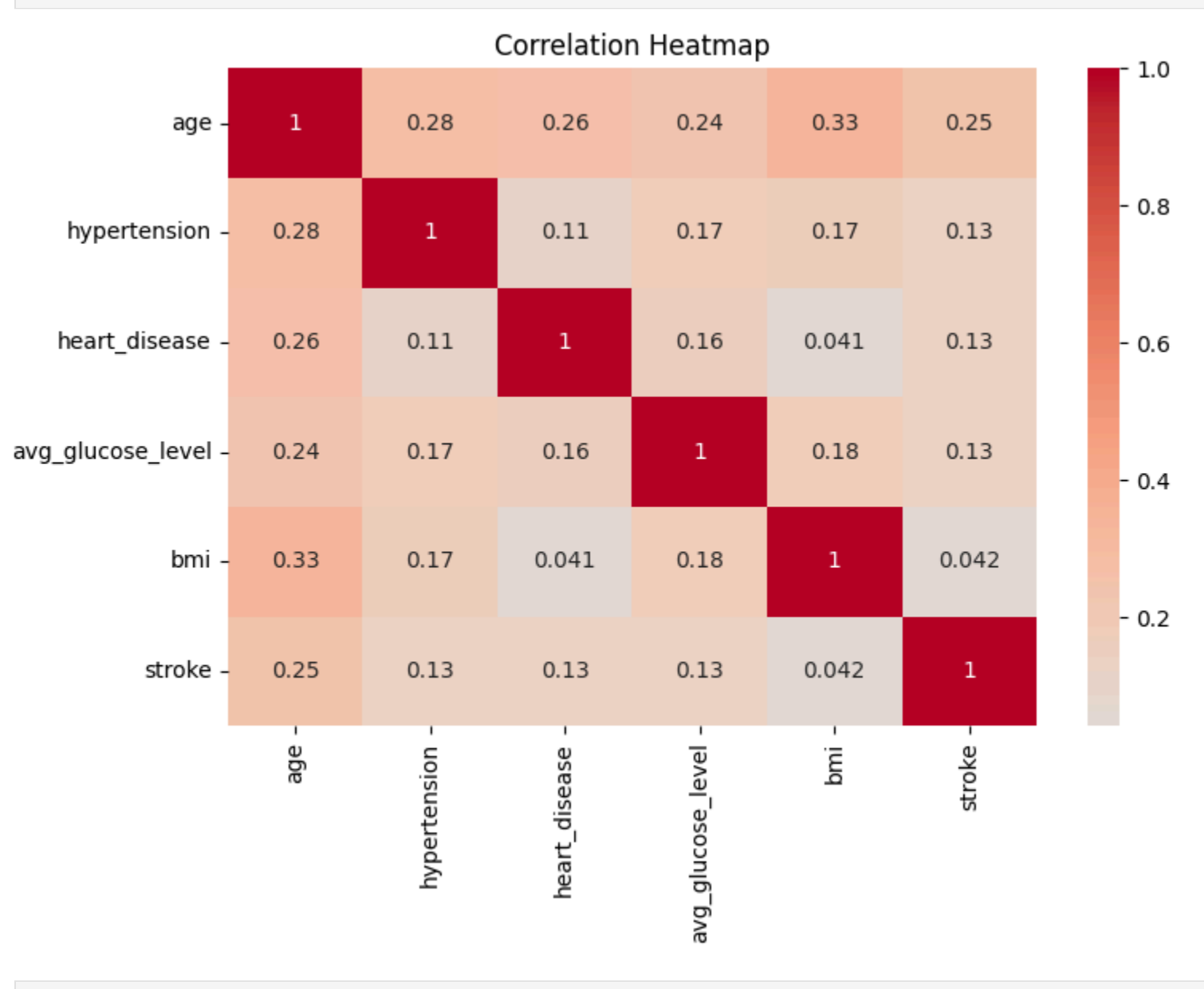
Out[37]:

stroke	0	1
smoking_status		
Unknown	96.955959	3.044041
formerly smoked	92.090395	7.909605
never smoked	95.243129	4.756871
smokes	94.676806	5.323194

In [38]:
```
fig, axes = plt.subplots(1, 2, figsize=(12,5))
sns.histplot(df, x='age', hue='stroke', bins=20, kde=True, ax=axes[0])
sns.histplot(df, x='bmi', hue='stroke', bins=20, kde=True, ax=axes[1])
axes[0].set_title('Age Distribution by Stroke')
axes[1].set_title('BMI Distribution by Stroke')
plt.tight_layout()
plt.show()
```



In [39]:
```
# Correlation (numeric only)
num_df = df.select_dtypes('number').copy()
if 'id' in num_df.columns:
    num_df = num_df.drop(columns=['id'])

corr = num_df.corr()

plt.figure(figsize=(8,6))
sns.heatmap(corr, cmap='coolwarm', center=0, annot=True)
plt.title('Correlation Heatmap')
plt.tight_layout()
plt.show()
```



In [40]:
```
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 sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
```

In [41]:
```
# Drop irrelevant ID column if present
if 'id' in df.columns:
    df = df.drop(columns=['id'])

# Handle missing BMI
if 'bmi' in df.columns:
    df['bmi'] = df['bmi'].fillna(df['bmi'].median())

# Split features/target
X = df.drop('stroke', axis=1)
y = df['stroke']

# Identify categorical & numeric columns
cat_cols = X.select_dtypes('object').columns.tolist()
num_cols = X.select_dtypes(exclude='object').columns.tolist()

print("Categorical:", cat_cols)
print("Numeric:", len(num_cols), "columns")
```

```
Categorical: ['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status']
Numeric: 5 columns
```

In [42]:
```
# Build preprocessor
preprocessor = ColumnTransformer([
    ('num', Pipeline([
        ('imputer', SimpleImputer(strategy='median')),
        ('scaler', StandardScaler())
    ]), num_cols),
    ('cat', Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('encoder', OneHotEncoder(handle_unknown='ignore'))
    ]), cat_cols)
])

# Split data (stratify to preserve class balance)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=42, stratify=y
)
```

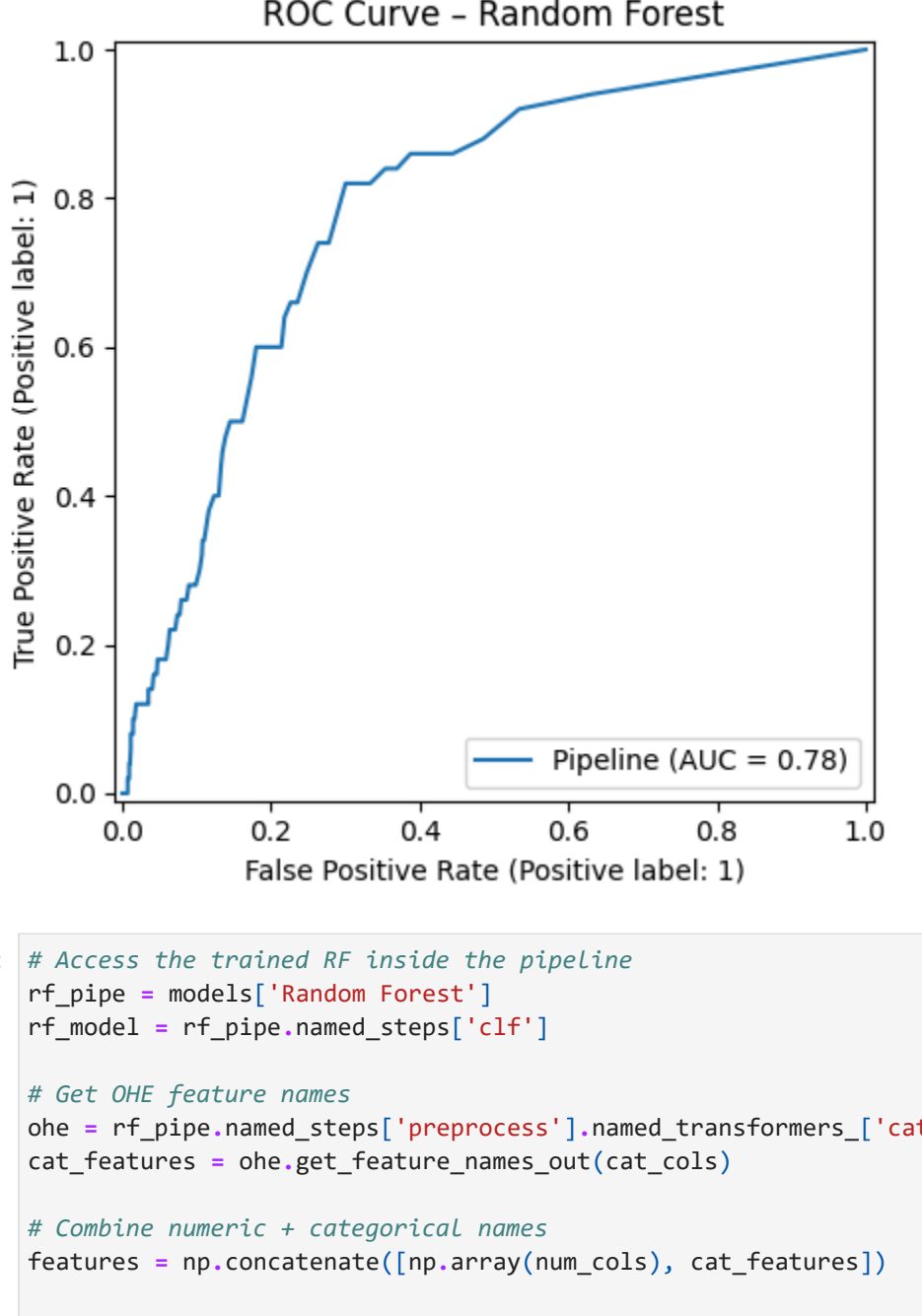
In [43]:
```
# Logistic Regression
logreg = Pipeline([
    ('preprocess', preprocessor),
    ('clf', LogisticRegression(max_iter=500, class_weight='balanced'))
])

# Random Forest
rf = Pipeline([
    ('preprocess', preprocessor),
    ('clf', RandomForestClassifier(n_estimators=300, random_state=42,
                                   class_weight='balanced_subsample', n_jobs=-1))
])

models = {'Logistic Regression': logreg, 'Random Forest': rf}

for name, model in models.items():
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    print(f"\n===== {name} =====")
    print(classification_report(y_test, preds, digits=3))
    print("Confusion Matrix:\n", confusion_matrix(y_test, preds))
```

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

           0      0.986     0.743     0.847       972
           1      0.138     0.800     0.235        50

    accuracy                          0.746      1022
   macro avg      0.562     0.771     0.541      1022
weighted avg      0.945     0.746     0.817      1022

Confusion Matrix:
 [[722 250]
 [ 10  40]]

===== Random Forest =====
              precision    recall  f1-score   support

           0      0.951     0.999     0.974       972
           1      0.000     0.000     0.000        50

    accuracy                          0.950      1022
   macro avg      0.476     0.499     0.487      1022
weighted avg      0.905     0.950     0.927      1022

Confusion Matrix:
 [[971   1]
 [ 50   0]]
```

In [44]:
```
for name, model in models.items():
    if hasattr(model, "predict_proba"):
        proba = model.predict_proba(X_test)[:, 1]
        auc = roc_auc_score(y_test, proba)
        print(f"{name} ROC-AUC: {auc:.3f}")
```

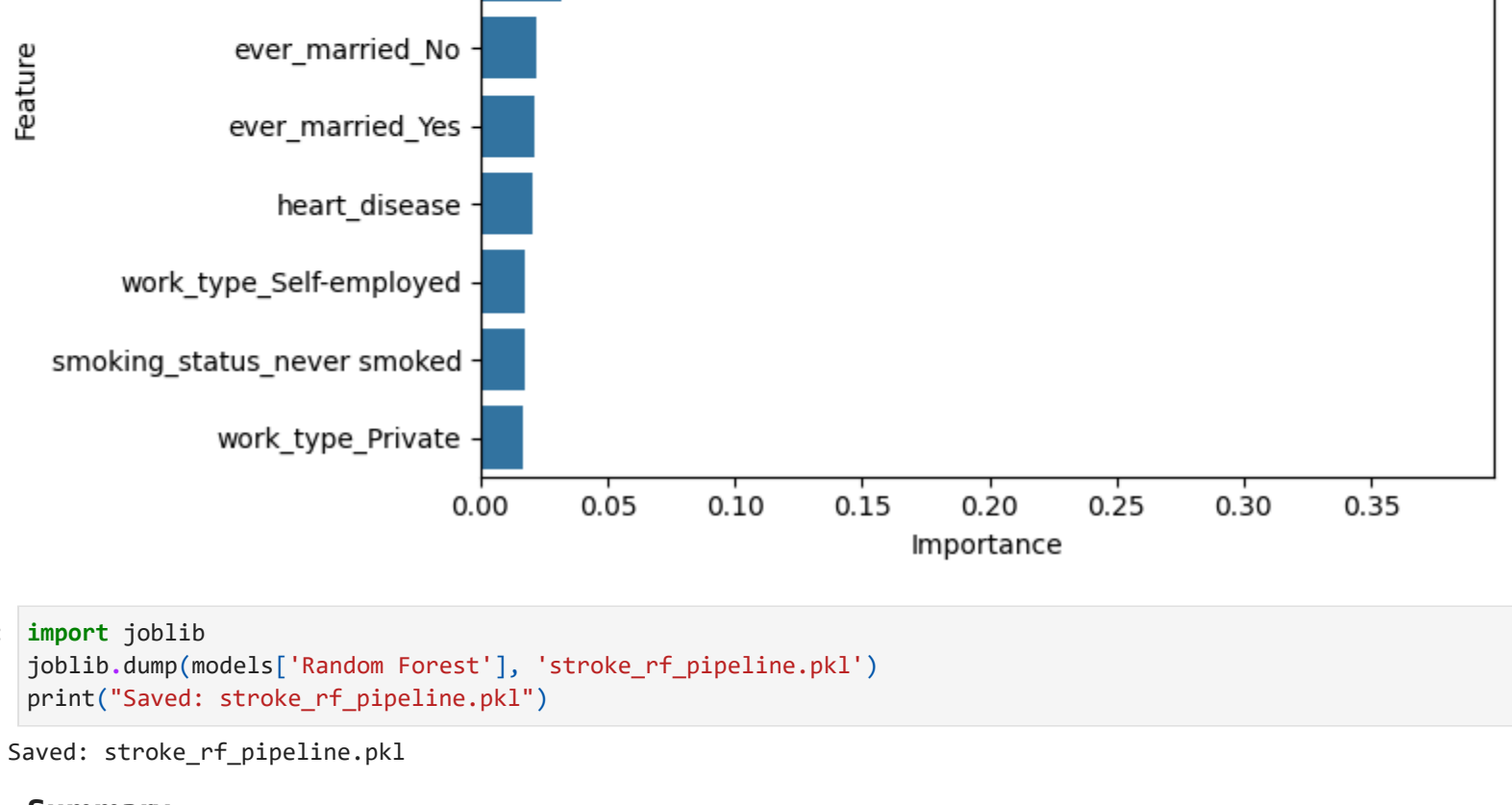
```
Logistic Regression ROC-AUC: 0.844
Random Forest ROC-AUC: 0.784
```

In [46]:
```
from sklearn.metrics import RocCurveDisplay

RocCurveDisplay.from_estimator(models['Random Forest'], X_test, y_test)
plt.title('ROC Curve – Random Forest')
plt.tight_layout(); plt.show()
```



In [45]:
```
# Access the trained RF inside the pipeline
rf_pipe = models['Random Forest']
rf_model = rf_pipe.named_steps['clf']

# Get OHE feature names
ohe = rf_pipe.named_steps['preprocess'].named_transformers_['cat'].named_steps['encoder']
cat_features = ohe.get_feature_names_out(cat_cols)

# Combine numeric + categorical names
features = np.concatenate([np.array(num_cols), cat_features])

# Feature importances
importances = rf_model.feature_importances_
feat_df = pd.DataFrame({'Feature': features, 'Importance': importances}) \
          .sort_values('Importance', ascending=False)

display(feat_df.head(10))

plt.figure(figsize=(8,5))
sns.barplot(x='Importance', y='Feature', data=feat_df.head(10))
plt.title('Top 10 Important Features (Random Forest)')
plt.tight_layout()
plt.show()
```

	Feature	Importance
0	age	0.379420
3	avg_glucose_level	0.176162
4	bmi	0.169755
1	hypertension	0.031563
8	ever_married_No	0.022191
9	ever_married_Yes	0.021066
2	heart_disease	0.020836
13	work_type_Self-employed	0.017444
19	smoking_status_never smoked	0.017086
12	work_type_Private	0.016524



In [47]:
```
import joblib
joblib.dump(models['Random Forest'], 'stroke_rf_pipeline.pkl')
print("Saved: stroke_rf_pipeline.pkl")
```

```
Saved: stroke_rf_pipeline.pkl
```

Summary

- The dataset showed a strong class imbalance, with most records labeled as non-stroke.
- Age and average glucose level exhibited clear positive correlations with stroke occurrence, particularly for individuals aged 60–80 or with glucose levels above 125 mg/dL.
- Gender showed no significant effect, while hypertension and heart disease were more common among stroke cases.

Methodology

- Built preprocessing pipeline using ColumnTransformer:
- Median imputation for missing BMI values
- Standard scaling for numerical features
- One-hot encoding for categorical variables
- Trained and evaluated two baseline models:
- Logistic Regression with class balancing
- Random Forest with balanced subsampling

Results (Test Set)

- Logistic Regression: ROC-AUC = 0.844, recall = 0.80, best at detecting stroke cases.

Random Forest: ROC-AUC = 0.784, high overall accuracy but struggled with minority class (stroke = 1).

Key Predictive Features

- age, avg_glucose_level, hypertension, heart_disease, and bmi ranked highest in feature importance, aligning with known medical risk factors.

Conclusion

The model highlights the trade-off between accuracy and recall when dealing with imbalanced medical datasets. Disclaimer: This is an educational project; correlation does not imply causation, and dataset size and bias should be considered.