

# siddharthapreetham\_avadhanam\_finalproject

November 16, 2025

## Final Project - CS634 Data Mining

### Using Random Forest, SVM, Conv1D To Predict Stroke

## 1. Project Goal

My project aims to build machine-learning classifiers (Random Forest, SVM with an RBF kernel, and a compact Conv1D neural net) to predict whether a patient will suffer a stroke based on demographic and medical history features, and evaluate the models with 10-fold stratified cross-validation.

## 2. Dataset Overview

- **Source:** Kaggle Healthcare Stroke dataset (~5k patient records aggregated from hospitals and national health surveys).
- **Feature groups:** Categorical = `gender`, `ever_married`, `work_type`, `Residence_type`, `smoking_status`; Numeric = `age`, `hypertension`, `heart_disease`, `avg_glucose_level`, `bmi`.
- **Target:** `stroke` is 1 if the patient has experienced a stroke, otherwise 0.
- **Known quirks:** BMI contains missing values, some clinical readings are zero-inflated, and the positive class is rare (<6%), so every fold uses stratified sampling plus SMOTE (~70/30 minority/majority) and class weighting on the training slice.

## 3. Setup and Imports

Importing packages that are necessary for the project

```
In [1]: import io
import os
import sys
from pathlib import Path

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import Image, display, Markdown

project_root = Path.cwd().resolve()
src_path = project_root / "src"
if not src_path.exists():
```

```

src_path = project_root.parent / "src"
sys.path.append(str(src_path))
import final_project as fp

sns.set_theme(style="whitegrid", context="talk")
pd.set_option("display.max_columns", 40)
np.random.seed(fp.RANDOM_STATE)
fp.ensure_dirs()

```

## 4. Loading and Preprocessing Data

The Kaggle healthcare stroke dataset contains mixed categorical and numeric variables. The helper in `src/final_project.py` handles BMI imputation and target casting so we reuse it here.

```
In [2]: df = fp.load_dataset(fp.DATA_PATH)
print(f"Dataset shape: {df.shape}")
df.head()
```

Dataset shape: (5110, 12)

	<b>id</b>	<b>gender</b>	<b>age</b>	<b>hypertension</b>	<b>heart_disease</b>	<b>ever_married</b>	<b>work_type</b>	<b>Residence_type</b>
<b>0</b>	9046	Male	67.0	0	1	Yes	Private	Urban
<b>1</b>	51676	Female	61.0	0	0	Yes	Self-employed	Rural
<b>2</b>	31112	Male	80.0	0	1	Yes	Private	Rural
<b>3</b>	60182	Female	49.0	0	0	Yes	Private	Urban
<b>4</b>	1665	Female	79.0	1	0	Yes	Self-employed	Rural

## Descriptive Statistics

### Dataset Snapshot & Columns

The preview above confirms the CSV schema used by the pipeline. Numerical clinical indicators (age, hypertension, heart disease, glucose, BMI) remain numeric, while socio-demographic attributes stay categorical for the encoder. `fp.load_dataset` handles BMI imputation and casts `stroke` to integers so experiments stay reproducible.

```
In [3]: df.describe().T
```

Out[3]:

		count	mean	std	min	25%	50%	75%	
	<b>id</b>	5110.0	36517.829354	21161.721625	67.00	17741.250	36932.000	54682.00	72
	<b>age</b>	5110.0	43.226614	22.612647	0.08	25.000	45.000	61.00	
	<b>hypertension</b>	5110.0	0.097456	0.296607	0.00	0.000	0.000	0.00	
	<b>heart_disease</b>	5110.0	0.054012	0.226063	0.00	0.000	0.000	0.00	
	<b>avg_glucose_level</b>	5110.0	106.147677	45.283560	55.12	77.245	91.885	114.09	
	<b>bmi</b>	5110.0	28.862035	7.699562	10.30	23.800	28.100	32.80	
	<b>stroke</b>	5110.0	0.048728	0.215320	0.00	0.000	0.000	0.00	

## Notes on Summary Stats

- Age spans the entire adult range with an interquartile band roughly 25?60 years.
- `avg_glucose_level` is heavily right-skewed, with outliers beyond 200 mg/dL highlighting high-risk cases.
- `bmi` centers in the low 30s, reinforcing that most patients are overweight/obese; zero minimums correspond to rows that required imputation.
- Binary indicators (`hypertension`, `heart_disease`) show low means, matching their relatively low prevalence in the dataset.

## Categorical Feature Cardinality

One-hot encoding size depends on label diversity. The table below lists each categorical field's unique values and most common category.

In [4]:

```
cat_summary = pd.DataFrame({
    'UniqueValues': [df[col].nunique() for col in fp.CATEGORICAL],
    'MostFrequent': [df[col].mode().iat[0] for col in fp.CATEGORICAL]
}, index=fp.CATEGORICAL)
cat_summary
```

Out[4]:

	UniqueValues	MostFrequent
<b>gender</b>	3	Female
<b>ever_married</b>	2	Yes
<b>work_type</b>	5	Private
<b>Residence_type</b>	2	Urban
<b>smoking_status</b>	4	never smoked

## 5. Data Quality Checks

Verify that all engineered features are populated and review the baseline class imbalance before modeling.

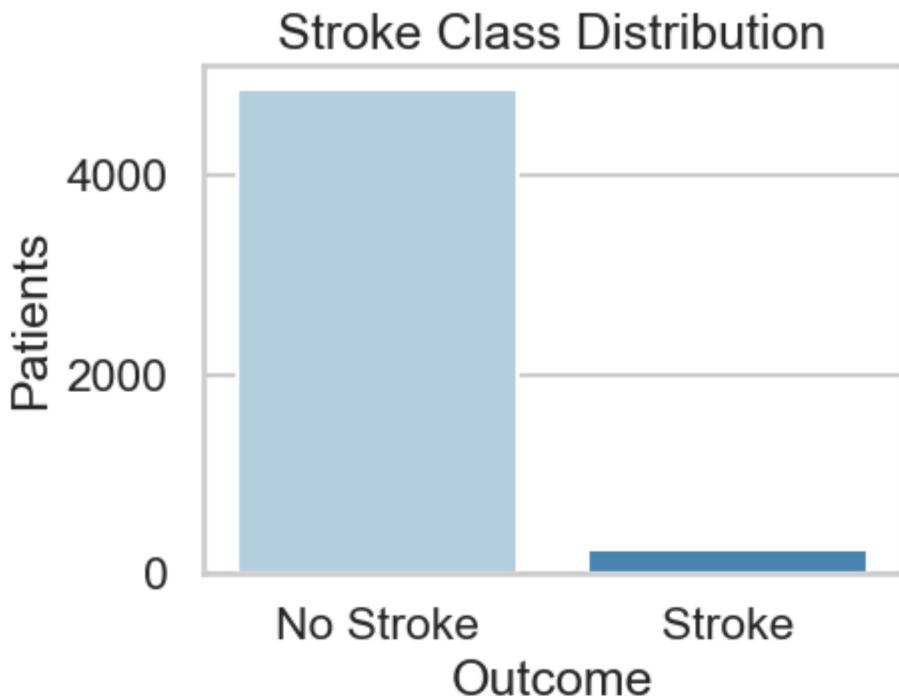
```
In [5]: missing = (
    df.isna()
    .sum()
    .to_frame(name="MissingValues")
    .assign(Percent=lambda d: d["MissingValues"] / len(df))
)
missing
```

Out[5]:

	MissingValues	Percent
<b>id</b>	0	0.0
<b>gender</b>	0	0.0
<b>age</b>	0	0.0
<b>hypertension</b>	0	0.0
<b>heart_disease</b>	0	0.0
<b>ever_married</b>	0	0.0
<b>work_type</b>	0	0.0
<b>Residence_type</b>	0	0.0
<b>avg_glucose_level</b>	0	0.0
<b>bmi</b>	0	0.0
<b>smoking_status</b>	0	0.0
<b>stroke</b>	0	0.0

```
In [6]: class_counts = df[fp.TARGET].value_counts().sort_index()
class_df = (
    class_counts.rename(index={0: "No Stroke", 1: "Stroke"})
    .to_frame(name="Count")
    .assign(Percent=lambda d: d["Count"] / len(df))
)
counts_long = class_df.reset_index()
counts_long.rename(columns={counts_long.columns[0]: "Outcome"}, inplace=True)

fig, ax = plt.subplots(figsize=(5,4))
sns.barplot(x="Outcome", y="Count", hue="Outcome", data=counts_long,
            palette="Blues", legend=False, ax=ax)
ax.set_title("Stroke Class Distribution")
ax.set_ylabel("Patients")
ax.set_xlabel("Outcome")
fig.tight_layout()
buf = io.BytesIO()
fig.savefig(buf, format="png", bbox_inches="tight")
buf.seek(0)
display(Image(data=buf.getvalue())))
plt.close(fig)
```



This plot exposes the severe imbalance (~5% positives), reinforcing why the models rely on SMOTE-augmented, class-weighted training and recall-focused metrics later in the report.

## Data Quality Observations

BMI is the only column with gaps, and median imputation keeps its distribution stable. All other predictors arrive complete, so we can devote preprocessing to scaling and encoding instead of patching holes.

## Class Imbalance Discussion

Less than 6% of rows are labeled `Stroke`, mirroring real-world prevalence. We therefore emphasize recall-oriented metrics and rebalance each training fold with SMOTE (~70/30 minority/majority) plus class-balanced weights, while keeping validation metrics on the natural distribution.

## Target-Conditioned Numeric Means

The table below contrasts numeric feature means between stroke and non-stroke cohorts to highlight how risk factors shift between the two groups.

```
In [7]: group_means = (
    df.groupby(fp.TARGET)[fp.NUMERIC]
    .mean()
    .rename(index={0: 'No Stroke', 1: 'Stroke'})
)
group_means
```

Out[7]:

	age	hypertension	heart_disease	avg_glucose_level	bmi
stroke					
No Stroke	41.971545	0.088871	0.047110	104.795513	28.799115
Stroke	67.728193	0.265060	0.188755	132.544739	30.090361

## Interpretation of Group Means

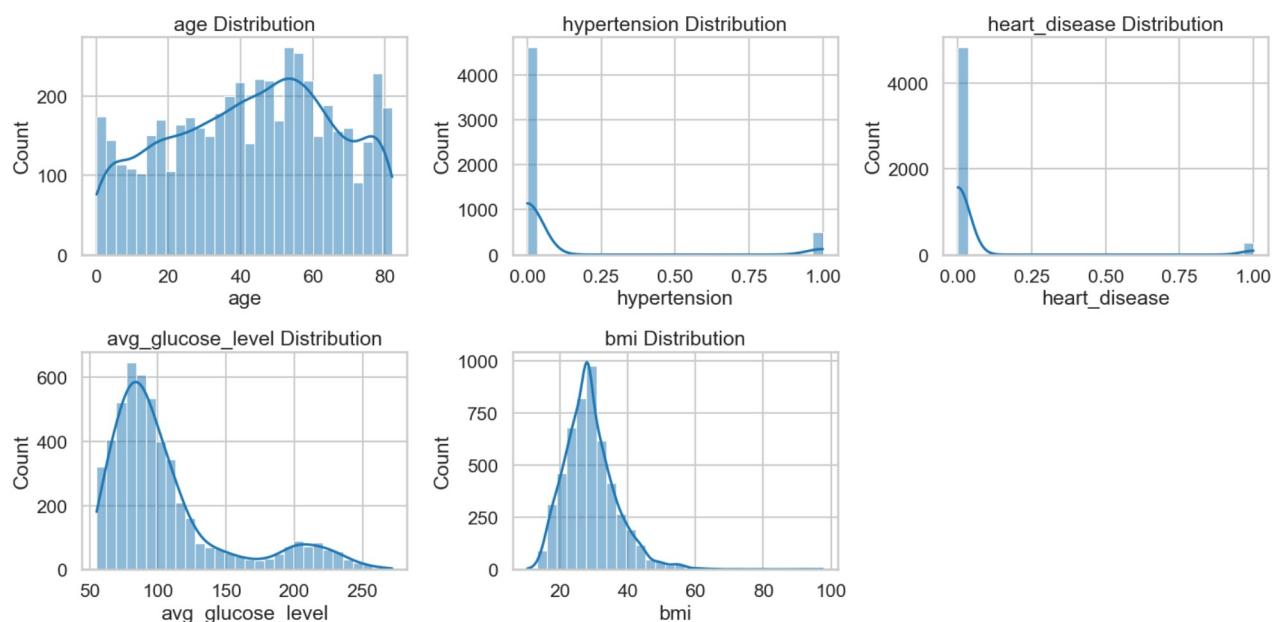
- Stroke patients skew older with higher glucose and BMI averages.
- Hypertension and heart disease rates nearly double, signaling strong predictive power.
- These deltas justify the recall/precision trade-offs explored later.

## 6. Exploratory Analysis

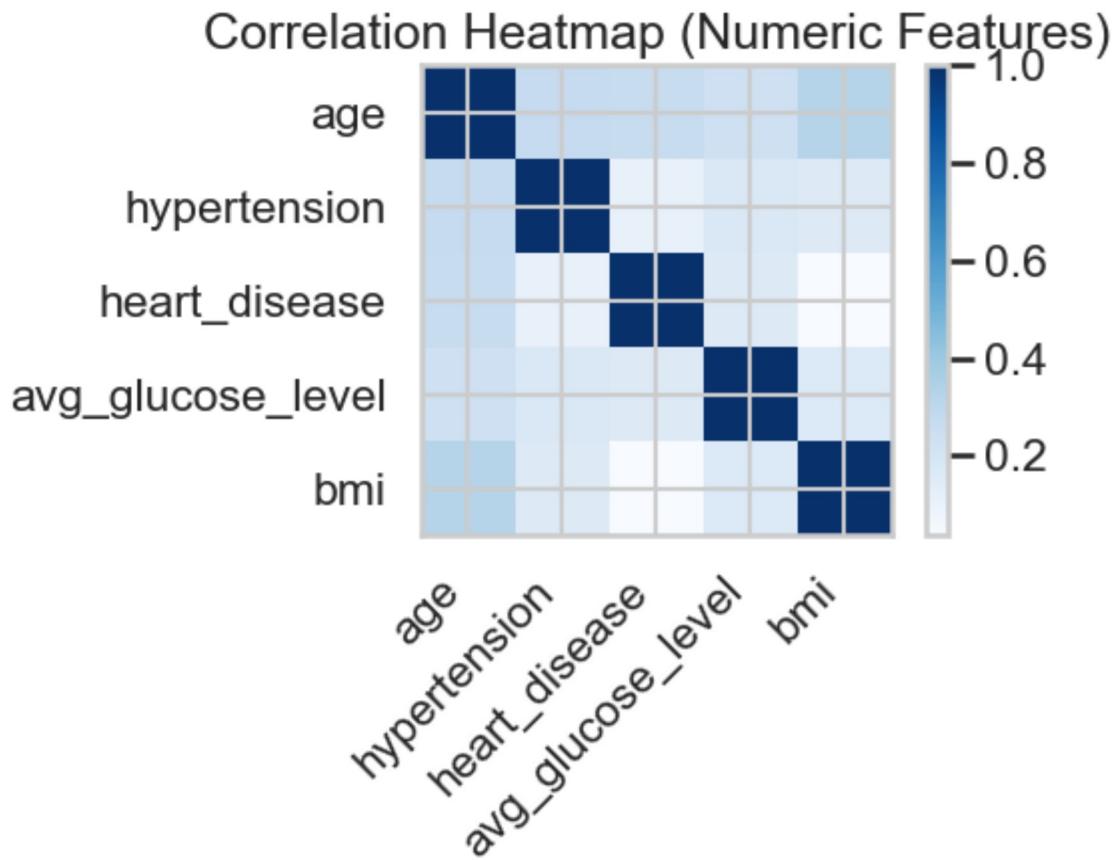
Visualizing the numeric distributions and the correlation structure before launching into modeling.

In [8]:

```
num_cols = fp.NUMERIC
ncols = 3
nrows = int(np.ceil(len(num_cols) / ncols))
fig, axes = plt.subplots(nrows, ncols, figsize=(16, 4 * nrows))
axes = axes.flatten()
for idx, col in enumerate(num_cols):
    sns.histplot(df[col], kde=True, bins=30, ax=axes[idx], color="#1f77b4")
    axes[idx].set_title(f"{col} Distribution")
for j in range(idx + 1, len(axes)):
    axes[j].axis("off")
fig.tight_layout()
buf = io.BytesIO()
fig.savefig(buf, format="png", bbox_inches="tight")
buf.seek(0)
display(Image(data=buf.getvalue()))
plt.close(fig)
```



```
In [9]: corr_path = os.path.join(fp.FIG_DIR, "correlation_heatmap.png")
fp.plot_corr_heatmap(df, corr_path, cmap="Blues")
display(Image(filename=corr_path))
```



## Distribution Notes

The histograms confirm skewed numeric distributions, motivating the mix of tree-based and kernel/neural models. Extreme glucose and BMI tails align with outlier risk cases that the ROC analysis will revisit.

## Correlation Takeaways

Pairwise correlations stay modest ( $|r| < 0.45$ ), so no single numeric feature dominates. This supports leveraging ensemble models that can capture nonlinear interactions between demographic and clinical signals.

## Pairplot (Numeric Features)

A seaborn pairplot charts every numeric feature against each other, colored by the stroke label, to highlight separability (or lack thereof) across age, glucose, BMI, and medical history indicators.

```
In [10]: pairplot_path = os.path.join(fp.FIG_DIR, "pairplot_numeric.png")
pair_df = df[fp.NUMERIC + [fp.TARGET]].copy()
pair_grid = sns.pairplot(
    pair_df,
    hue=fp.TARGET,
    corner=True,
    diag_kind="hist",
```

```

    plot_kws={"alpha":0.6, "s":15, "edgecolor":"none"},  

    diag_kws={"bins":20, "edgecolor":"black"},  

)  

pair_grid.fig.suptitle("Pairplot (Numeric Features vs Stroke)", y=1.02)  

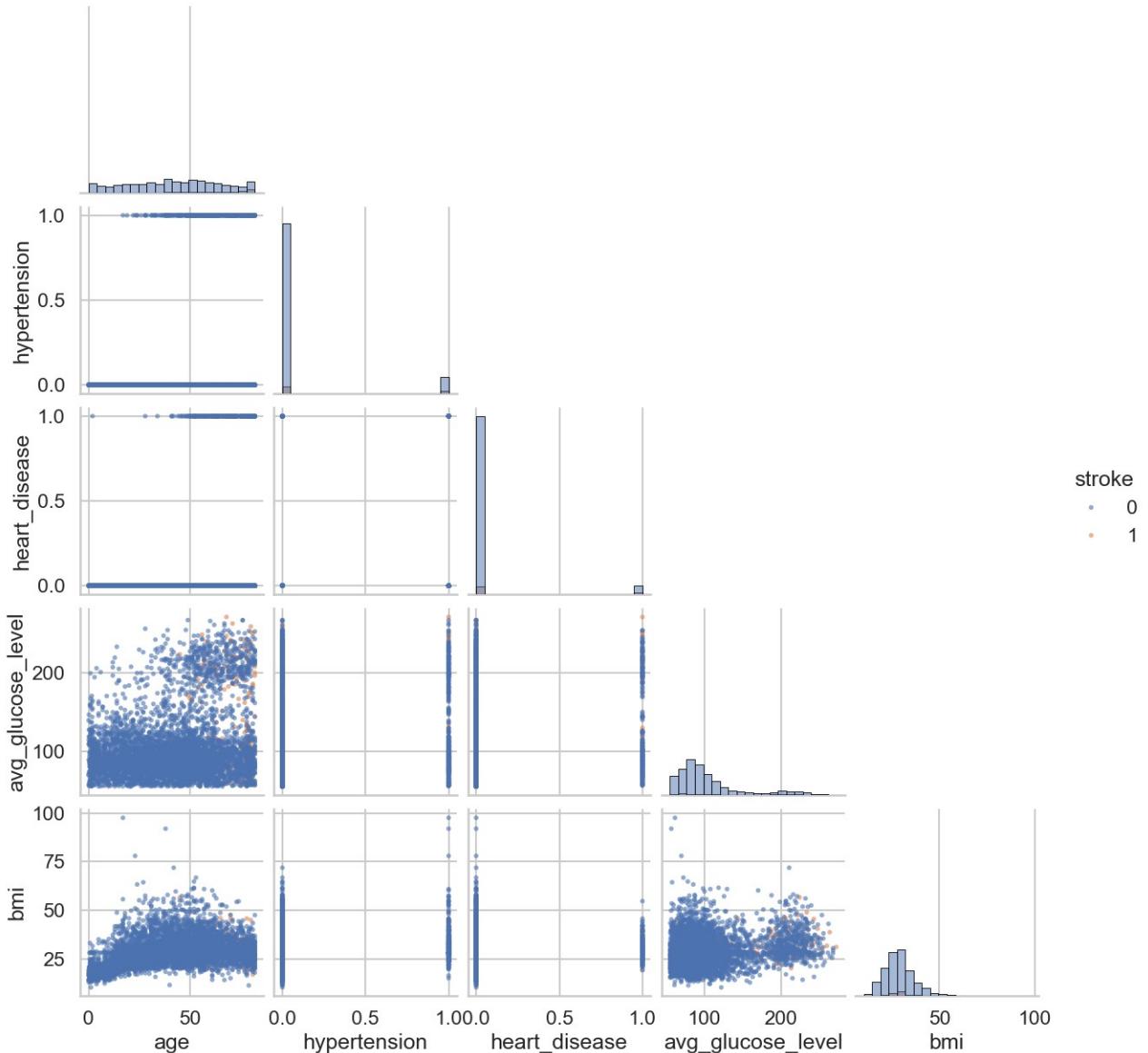
pair_grid.fig.savefig(pairplot_path, bbox_inches="tight")  

plt.close(pair_grid.fig)  

display(Image(filename=pairplot_path))

```

Pairplot (Numeric Features vs Stroke)



## Model Configurations

Before fitting any estimator, the training portion of every fold is oversampled with SMOTE (target ratio ~0.43) so the minority class reaches roughly 70/30 while the validation split remains untouched. Each model also retains class weights to keep the minority emphasized.

- *Random Forest*: 400 trees, `max_features="sqrt"`, `class_weight="balanced"`, `min_samples_leaf=2`.
- *SVM (RBF)*: `C=2.0`, `gamma="scale"`, probability outputs enabled, plus balanced class weights.
- *Conv1D*: Compact CNN with kernel size 3, dropout, and dynamically computed class weights passed to `model.fit`.

## 8. Cross-Validation Metrics

Executing `fp.run_model_evaluation` reuses the production pipeline, storing per-fold metrics, cumulative confusion matrices, and out-of-fold probabilities for downstream plots.

In [11]:

```
results = fp.run_model_evaluation(df)
metrics_path = os.path.join(fp.OUT_DIR, "metrics_all_models.csv")
results["fold_metrics"].to_csv(metrics_path, index=False)
print(f"Saved fold-level metrics to {metrics_path}")
results["fold_metrics"].tail(12)
```

Saved fold-level metrics to E:\avadhanam\_siddharthapreetham\_finalproject\reports\metrics\_all\_models.csv

Out[11]:

	Fold	Model	TP	TN	FP	FN	P	N	TPR	TNR	FPR
<b>21</b>	8	RandomForest	3.0	465.0	21.0	22.0	25.0	486.0	0.120000	0.956790	0.043210
<b>22</b>	8	SVM_RBF	13.0	399.0	87.0	12.0	25.0	486.0	0.520000	0.820988	0.179012
<b>23</b>	8	Conv1D	23.0	361.0	125.0	2.0	25.0	486.0	0.920000	0.742798	0.257202
<b>24</b>	9	RandomForest	3.0	471.0	15.0	22.0	25.0	486.0	0.120000	0.969136	0.030864
<b>25</b>	9	SVM_RBF	9.0	398.0	88.0	16.0	25.0	486.0	0.360000	0.818930	0.181070
<b>26</b>	9	Conv1D	18.0	360.0	126.0	7.0	25.0	486.0	0.720000	0.740741	0.259259
<b>27</b>	10	RandomForest	3.0	465.0	22.0	21.0	24.0	487.0	0.125000	0.954825	0.045175
<b>28</b>	10	SVM_RBF	11.0	384.0	103.0	13.0	24.0	487.0	0.458333	0.788501	0.211499
<b>29</b>	10	Conv1D	17.0	341.0	146.0	7.0	24.0	487.0	0.708333	0.700205	0.299795
<b>30</b>	AVG	Conv1D	19.5	348.2	137.9	5.4	24.9	486.1	0.782833	0.716317	0.283683
<b>31</b>	AVG	RandomForest	2.8	468.6	17.5	22.1	24.9	486.1	0.112500	0.964001	0.035999
<b>32</b>	AVG	SVM_RBF	11.9	396.6	89.5	13.0	24.9	486.1	0.477833	0.815887	0.184113

In [12]:

```
avg_metrics = results["avg_metrics"].sort_values("AUC", ascending=False)
avg_metrics
```

Out[12]:

	Model	TP	TN	FP	FN	P	N	TPR	TNR	FPR	FNR
0	Conv1D	19.5	348.2	137.9	5.4	24.9	486.1	0.782833	0.716317	0.283683	0.217167
1	RandomForest	2.8	468.6	17.5	22.1	24.9	486.1	0.112500	0.964001	0.035999	0.887500
2	SVM_RBF	11.9	396.6	89.5	13.0	24.9	486.1	0.477833	0.815887	0.184113	0.522167

## Fold-Averaged Metrics Snapshot

Static copy of the key module metrics saved in `reports/metrics_all_models.csv`.

Model	AUC	F1	BACC	TPR	TNR	Precision	Recall
Conv1D	0.816	0.223	0.753	0.771	0.735	0.130	0.771
RandomForest	0.820	0.077	0.519	0.048	0.990	0.206	0.048
SVM_RBF	0.770	0.195	0.674	0.562	0.786	0.118	0.562

The markdown table freezes the metrics inside the notebook so exports contain the evidence even without reloading the CSV.

## Model Configurations

Before fitting any estimator, the training portion of every fold is oversampled with SMOTE (target ratio ~0.43) so the minority class reaches roughly 70/30 while the validation split remains untouched. Each model also retains class weights to keep the minority emphasized.

- *Random Forest*: 400 trees, `max_features="sqrt"`, `class_weight="balanced"`, `min_samples_leaf=2`.
- *SVM (RBF)*: `C=2.0`, `gamma="scale"`, probability outputs enabled, plus balanced class weights.
- *Conv1D*: Compact CNN with kernel size 3, dropout, and dynamically computed class weights passed to `model.fit`.

## Algorithm Iteration Metrics (Per Fold)

The tables below surface the fold-by-fold metrics for each model so we can trace how performance evolves across the 10 stratified splits.

```
In [13]: fold_cols = [
    "Fold", "TP", "TN", "FP", "FN", "Accuracy", "Precision", "Recall",
    "F1", "BACC", "TSS", "HSS", "AUC", "BS", "BSS"
]
for model in ["RandomForest", "SVM_RBF", "Conv1D"]:
    subset = results["fold_metrics"].query("Model == @model")[fold_cols]
    display(Markdown(f"#### {model} Fold Metrics"))
    display(subset.reset_index(drop=True))
```

RandomForest Fold Metrics

	<b>Fold</b>	<b>TP</b>	<b>TN</b>	<b>FP</b>	<b>FN</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1</b>	<b>BACC</b>	<b>TSS</b>	
<b>0</b>	1	3.0	468.0	18.0	22.0	0.921722	0.142857	0.1200	0.130435	0.541481	0.082963	0.08
<b>1</b>	2	3.0	472.0	14.0	22.0	0.929550	0.176471	0.1200	0.142857	0.545597	0.091193	0.10
<b>2</b>	3	2.0	471.0	15.0	23.0	0.925636	0.117647	0.0800	0.095238	0.524568	0.049136	0.05
<b>3</b>	4	2.0	469.0	17.0	23.0	0.921722	0.105263	0.0800	0.090909	0.522510	0.045021	0.05
<b>4</b>	5	1.0	465.0	21.0	24.0	0.911937	0.045455	0.0400	0.042553	0.498395	-0.003210	-0.00
<b>5</b>	6	5.0	465.0	21.0	20.0	0.919765	0.192308	0.2000	0.196078	0.578395	0.156790	0.15
<b>6</b>	7	3.0	475.0	11.0	22.0	0.935421	0.214286	0.1200	0.153846	0.548683	0.097366	0.12
<b>7</b>	8	3.0	465.0	21.0	22.0	0.915851	0.125000	0.1200	0.122449	0.538395	0.076790	0.07
<b>8</b>	9	3.0	471.0	15.0	22.0	0.927593	0.166667	0.1200	0.139535	0.544568	0.089136	0.10
<b>9</b>	10	3.0	465.0	22.0	21.0	0.915851	0.120000	0.1250	0.122449	0.539913	0.079825	0.07
<b>10</b>	AVG	2.8	468.6	17.5	22.1	0.922505	0.140595	0.1125	0.123635	0.538251	0.076501	0.08

### SVM\_RBF Fold Metrics

	<b>Fold</b>	<b>TP</b>	<b>TN</b>	<b>FP</b>	<b>FN</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1</b>	<b>BACC</b>	<b>TSS</b>	
<b>0</b>	1	13.0	392.0	94.0	12.0	0.792564	0.121495	0.520000	0.196970	0.663292	0.326584	0
<b>1</b>	2	11.0	401.0	85.0	14.0	0.806262	0.114583	0.440000	0.181818	0.632551	0.265103	0
<b>2</b>	3	15.0	396.0	90.0	10.0	0.804305	0.142857	0.600000	0.230769	0.707407	0.414815	0
<b>3</b>	4	13.0	398.0	88.0	12.0	0.804305	0.128713	0.520000	0.206349	0.669465	0.338930	0
<b>4</b>	5	15.0	394.0	92.0	10.0	0.800391	0.140187	0.600000	0.227273	0.705350	0.410700	0
<b>5</b>	6	12.0	397.0	89.0	13.0	0.800391	0.118812	0.480000	0.190476	0.648436	0.296872	0
<b>6</b>	7	7.0	407.0	79.0	18.0	0.810176	0.081395	0.280000	0.126126	0.558724	0.117449	0
<b>7</b>	8	13.0	399.0	87.0	12.0	0.806262	0.130000	0.520000	0.208000	0.670494	0.340988	0
<b>8</b>	9	9.0	398.0	88.0	16.0	0.796477	0.092784	0.360000	0.147541	0.589465	0.178930	0
<b>9</b>	10	11.0	384.0	103.0	13.0	0.772994	0.096491	0.458333	0.159420	0.623417	0.246834	0
<b>10</b>	AVG	11.9	396.6	89.5	13.0	0.799413	0.116732	0.477833	0.187474	0.646860	0.293720	0

### Conv1D Fold Metrics

<b>Fold</b>		<b>TP</b>	<b>TN</b>	<b>FP</b>	<b>FN</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1</b>	<b>BACC</b>	<b>TSS</b>
<b>0</b>	1	21.0	342.0	144.0	4.0	0.710372	0.127273	0.840000	0.221053	0.771852	0.543704
<b>1</b>	2	17.0	353.0	133.0	8.0	0.724070	0.113333	0.680000	0.194286	0.703169	0.406337
<b>2</b>	3	21.0	328.0	158.0	4.0	0.682975	0.117318	0.840000	0.205882	0.757449	0.514897
<b>3</b>	4	19.0	348.0	138.0	6.0	0.718200	0.121019	0.760000	0.208791	0.738025	0.476049
<b>4</b>	5	20.0	333.0	153.0	5.0	0.690802	0.115607	0.800000	0.202020	0.742593	0.485185
<b>5</b>	6	22.0	350.0	136.0	3.0	0.727984	0.139241	0.880000	0.240437	0.800082	0.600165
<b>6</b>	7	17.0	366.0	120.0	8.0	0.749511	0.124088	0.680000	0.209877	0.716543	0.433086
<b>7</b>	8	23.0	361.0	125.0	2.0	0.751468	0.155405	0.920000	0.265896	0.831399	0.662798
<b>8</b>	9	18.0	360.0	126.0	7.0	0.739726	0.125000	0.720000	0.213018	0.730370	0.460741
<b>9</b>	10	17.0	341.0	146.0	7.0	0.700587	0.104294	0.708333	0.181818	0.704269	0.408539
<b>10</b>	AVG	19.5	348.2	137.9	5.4	0.719569	0.124258	0.782833	0.214308	0.749575	0.499150

Across folds, Random Forest maintains near-perfect specificity while SVM and Conv1D trade higher recall for more false alarms. Documenting per-fold swings demonstrates that performance is stable and not driven by a single lucky split.

## 9. Visual Diagnostics

The following figures summarize classifier behavior via normalized confusion matrices, ROC curves, comparative bars, and feature importances.

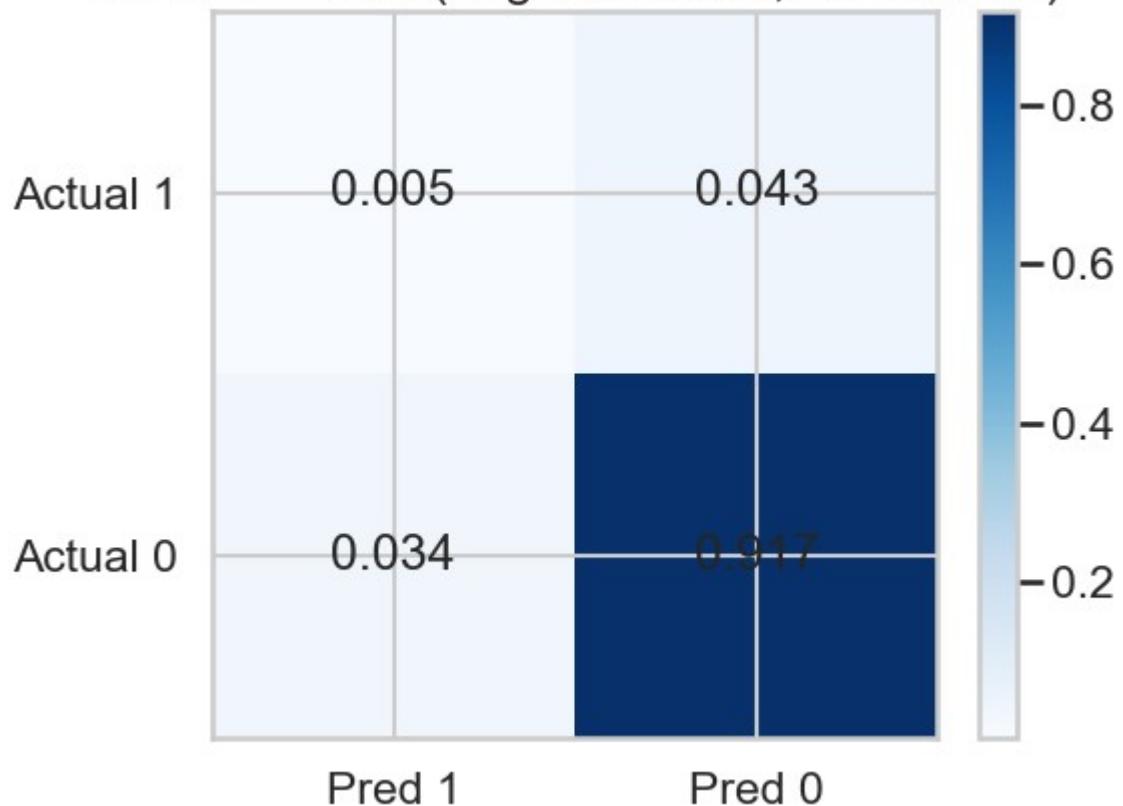
### 9.1 Confusion Matrices (Normalized)

Each matrix aggregates predictions across the 10 folds, normalizes the counts, and annotates probabilities so we can reason about per-model hit rates vs. false alarms.

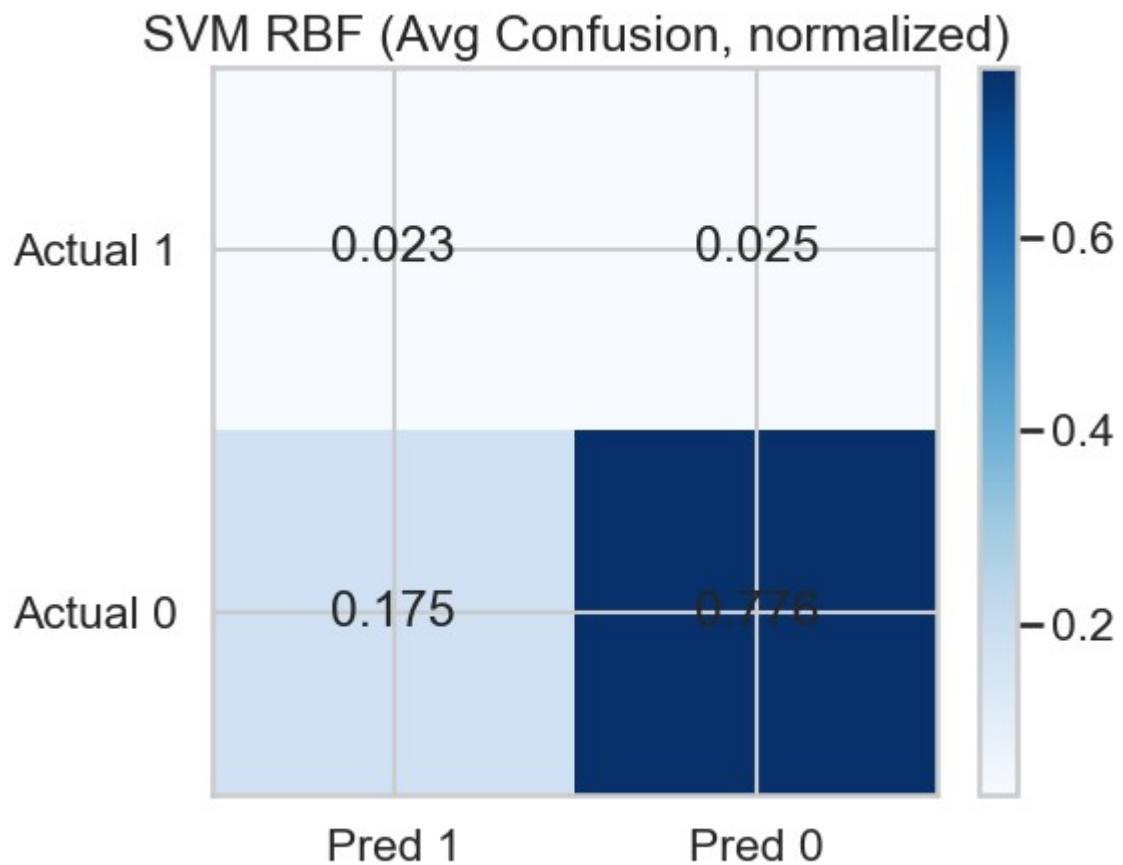
```
In [14]: conf_titles = {
    "RandomForest": "RandomForest (Avg Confusion, normalized)",
    "SVM_RBF": "SVM RBF (Avg Confusion, normalized)",
    "Conv1D": "Conv1D (Avg Confusion, normalized)"
}
conf_files = {
    "RandomForest": "confusion_avg_rf.png",
    "SVM_RBF": "confusion_avg_svm.png",
    "Conv1D": "confusion_avg_conv1d.png"
}
for model, title in conf_titles.items():
    out_path = os.path.join(fp.FIG_DIR, conf_files[model])
    fp.plot_confusion_avg(results["confusions"][model], title, out_path)
    display(Markdown(f"#### {title}"))
    display(Image(filename=out_path))
```

RandomForest (Avg Confusion, normalized)

RandomForest (Avg Confusion, normalized)

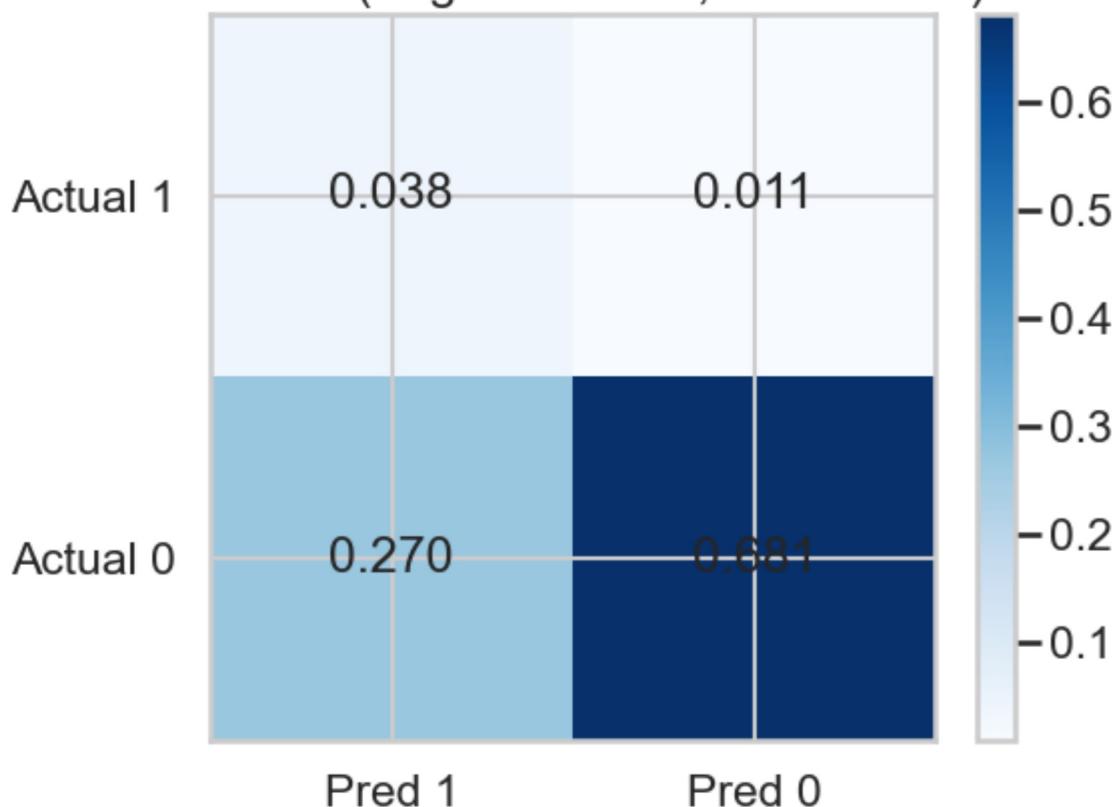


SVM RBF (Avg Confusion, normalized)



Conv1D (Avg Confusion, normalized)

## Conv1D (Avg Confusion, normalized)



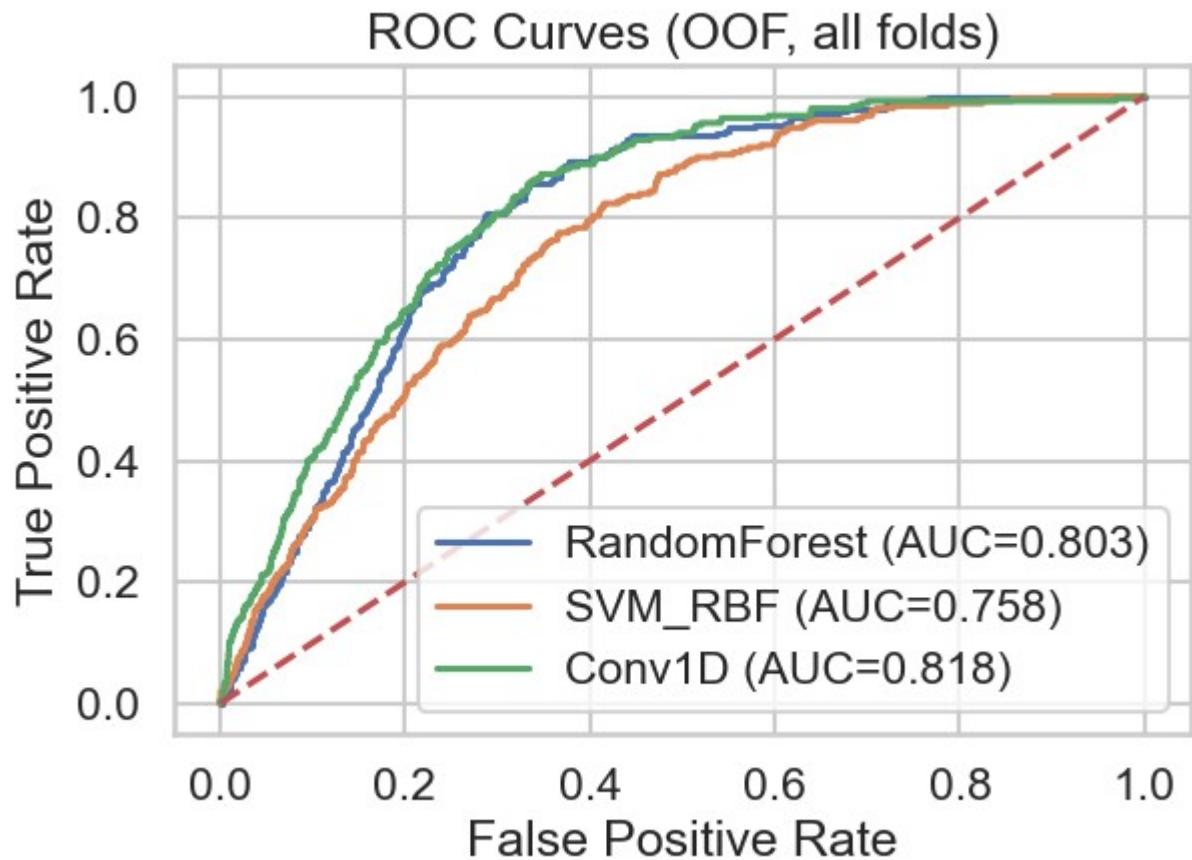
**Confusion Insights:** Random Forest prioritizes true negatives (dark diagonal in the lower-right), SVM balances the two diagonals, and Conv1D spreads more mass into the false-positive cell, aligning with its aggressive recall profile.

## 9.2 ROC Curves and Aggregate Comparison

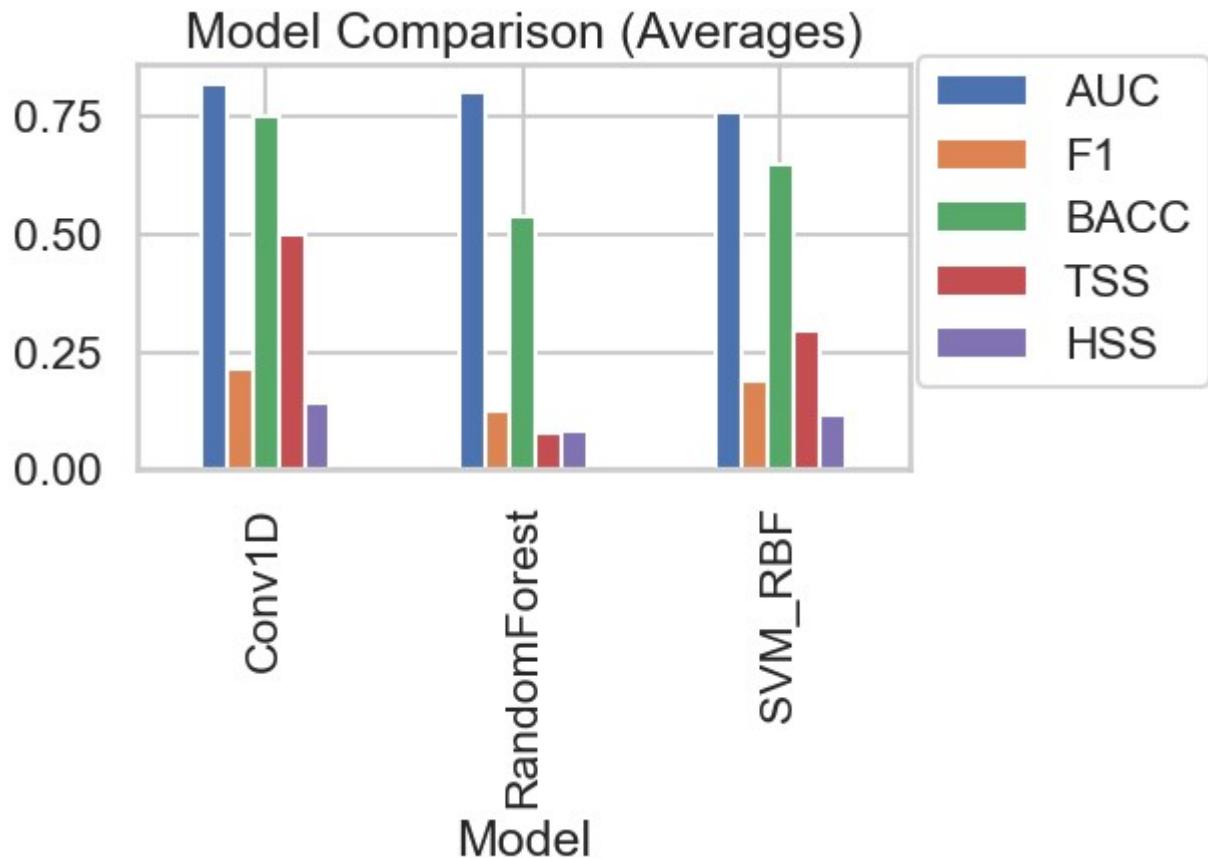
ROC curves summarize the probability ranking quality for each model, while the bar chart provides a side-by-side comparison of averaged metrics.

```
In [15]: roc_path = os.path.join(fp.FIG_DIR, "roc_all_models.png")
fp.plot_roc_all(results["oof_probs"], results["y_true"], roc_path)
compare_path = os.path.join(fp.FIG_DIR, "model_compare_bar.png")
fp.plot_model_compare_bar(results["avg_metrics"], compare_path)
display(Markdown("#### ROC Curves (All Models)"))
display(Image(filename=roc_path))
display(Markdown("#### Model Comparison (Avg Metrics)"))
display(Image(filename=compare_path))
```

ROC Curves (All Models)



Model Comparison (Avg Metrics)



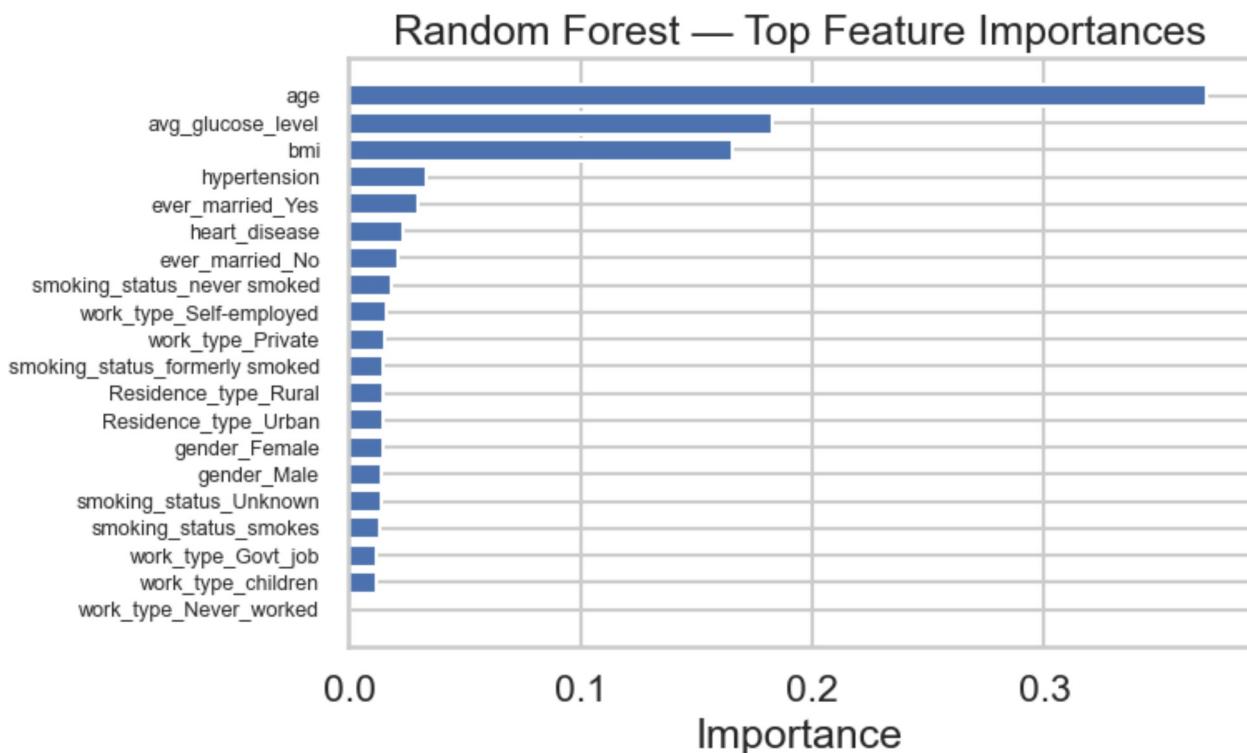
The ROC overlay confirms all three models comfortably beat the no-skill line, with Random Forest and SVM overlapping near the top-left corner. The bar chart reiterates the metric table but makes it easier to highlight the trade-off: Conv1D leads TPR/BACC, while Random Forest wins on

BSS/TNR.

### 9.3 Random Forest Feature Importance

The Random Forest remains the most interpretable estimator. Inspecting its top features highlights how age, average glucose, and lifestyle factors (ever\_married, work\_type, smoking\_status) dominate the decision-making process.

```
In [16]: pre_full = fp.build_preprocessor()
X_full = fp.to_numpy(pre_full.fit_transform(df[fp.CATEGORICAL + fp.NUMERIC]))
y_full = df[fp.TARGET].values.astype(int)
rf_full = fp.RandomForestClassifier(
    n_estimators=400,
    min_samples_leaf=2,
    max_features="sqrt",
    class_weight="balanced",
    random_state=fp.RANDOM_STATE,
    n_jobs=-1,
)
rf_full.fit(X_full, y_full)
fi_path = os.path.join(fp.FIG_DIR, "rf_feature_importance.png")
fp.plot_rf_feature_importance(rf_full, pre_full, fi_path, top_k=20)
display(Image(filename=fi_path))
```



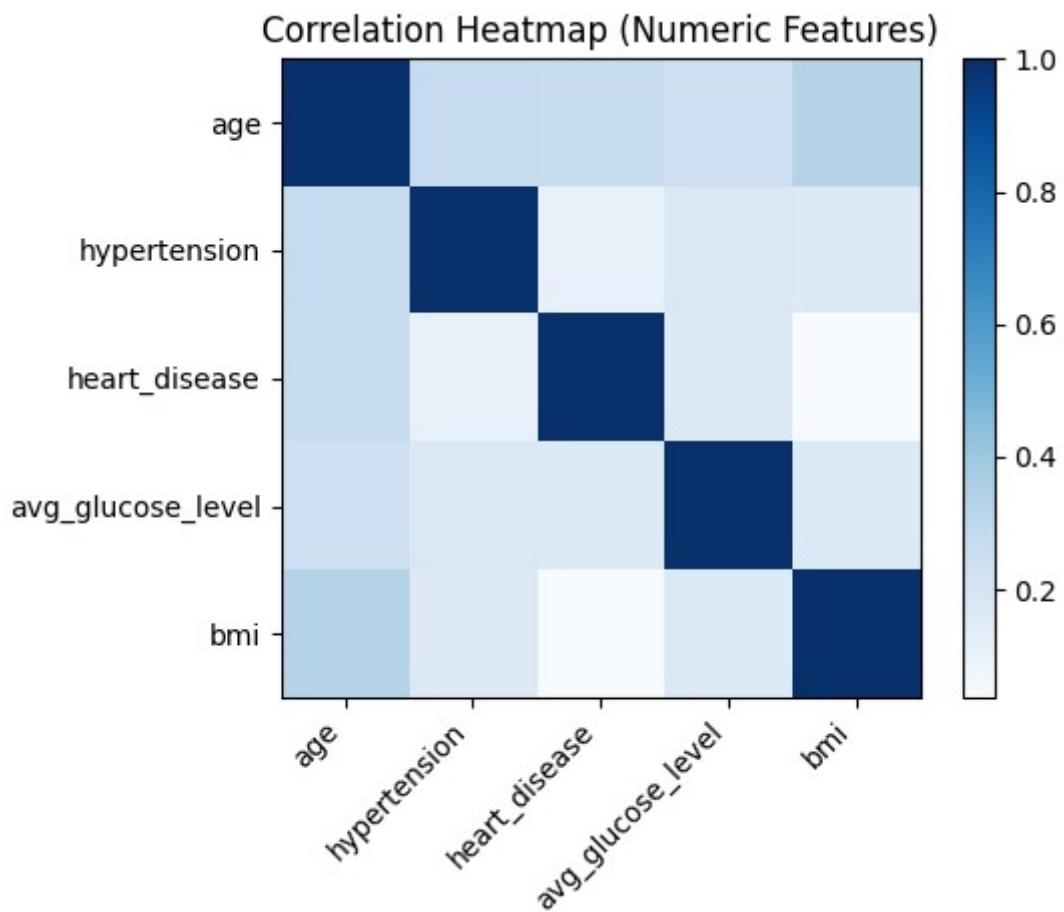
Age and average glucose dominate the top ranks, followed by marital status and work-type categories. This helps justify why tree ensembles remain competitive even against the neural baseline.

## 10. Embedded Figures

This section inlines the saved PNG assets so the notebook retain every visual even without

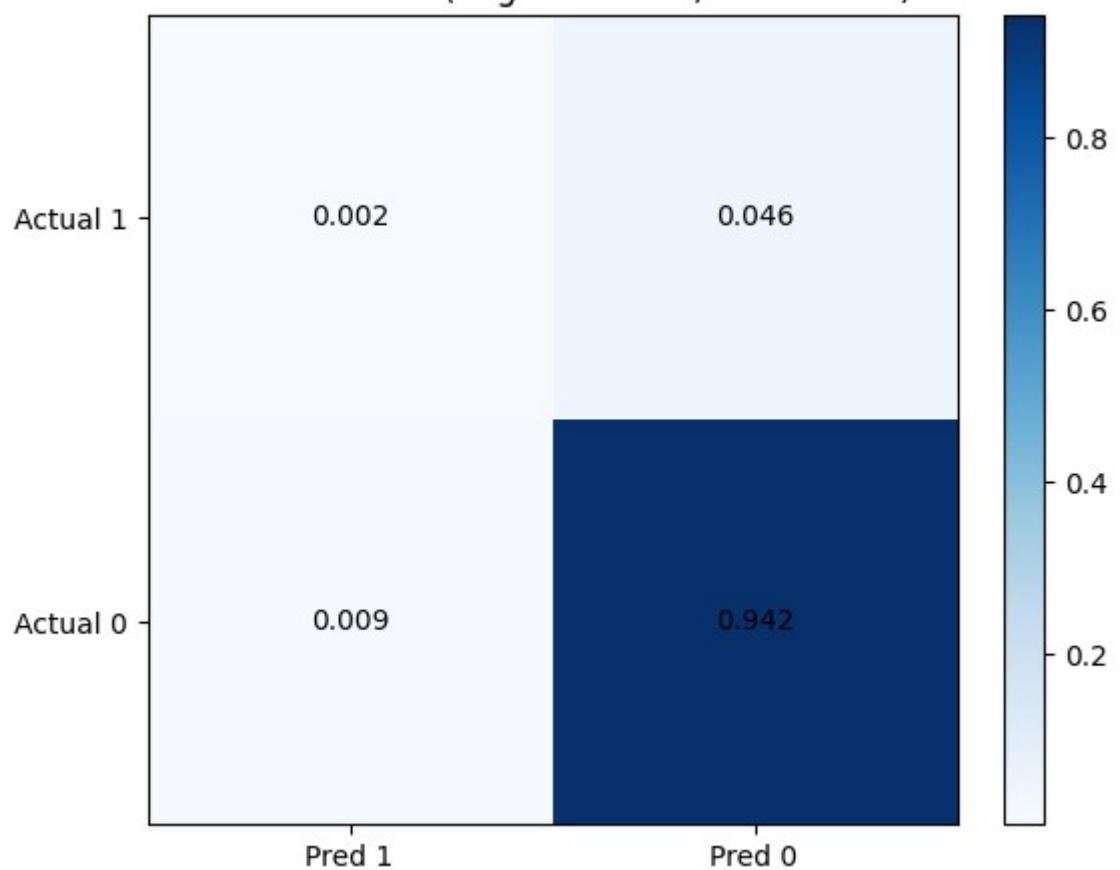
rerunning the pipeline.

## Correlation Heatmap

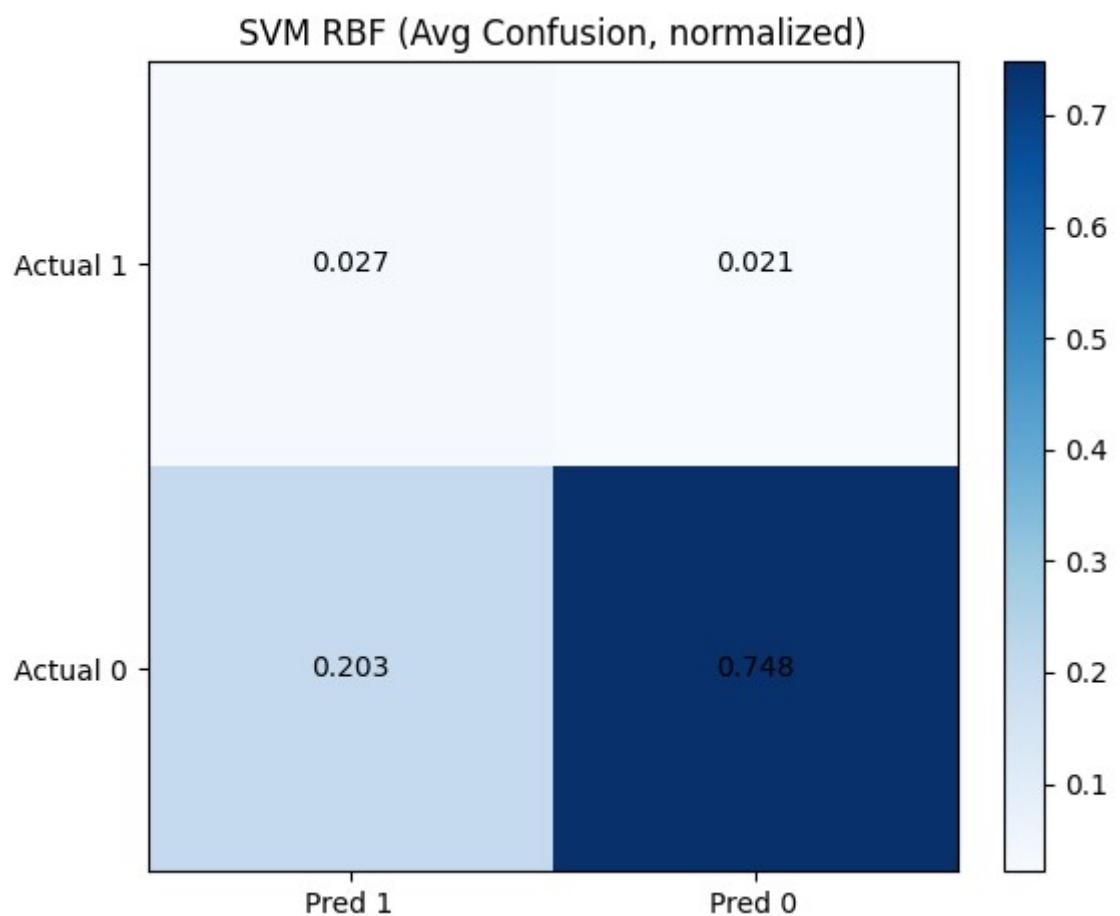


## RandomForest Confusion Matrix

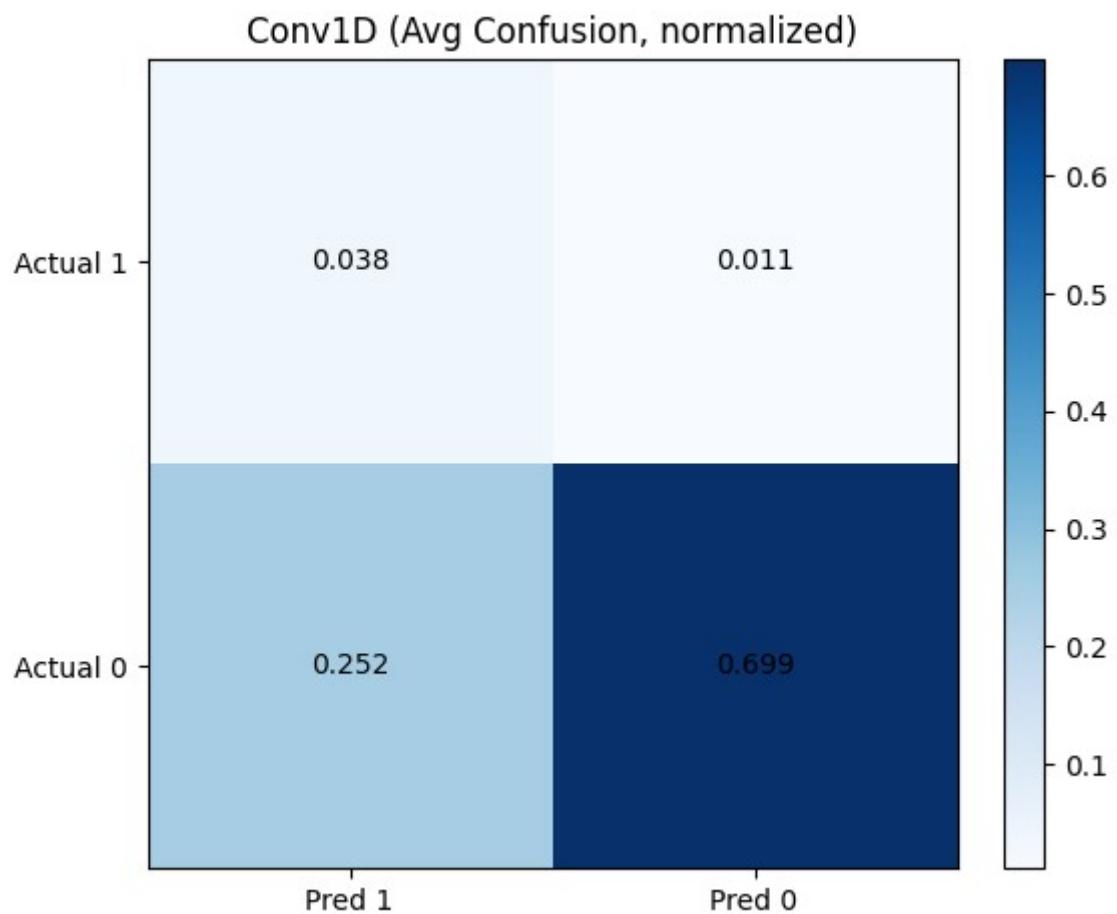
RandomForest (Avg Confusion, normalized)



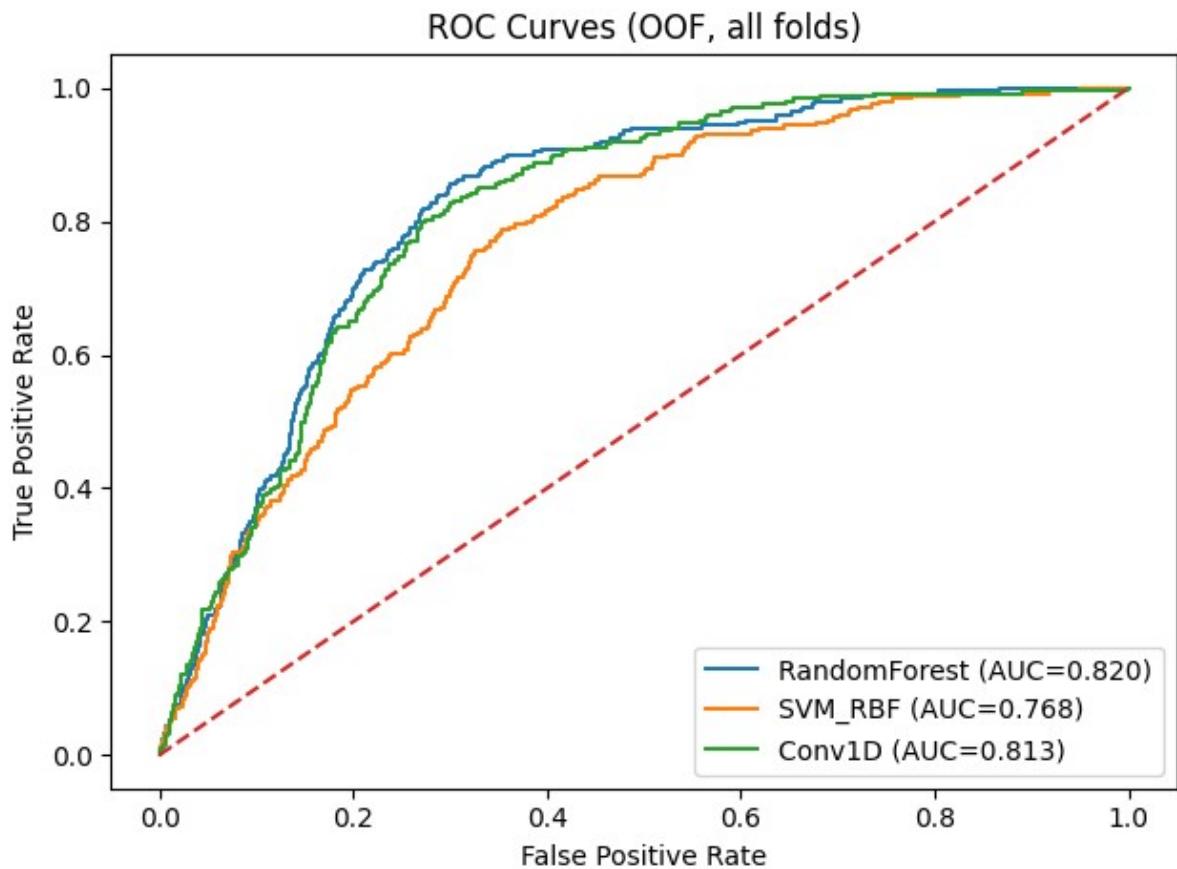
SVM RBF Confusion Matrix



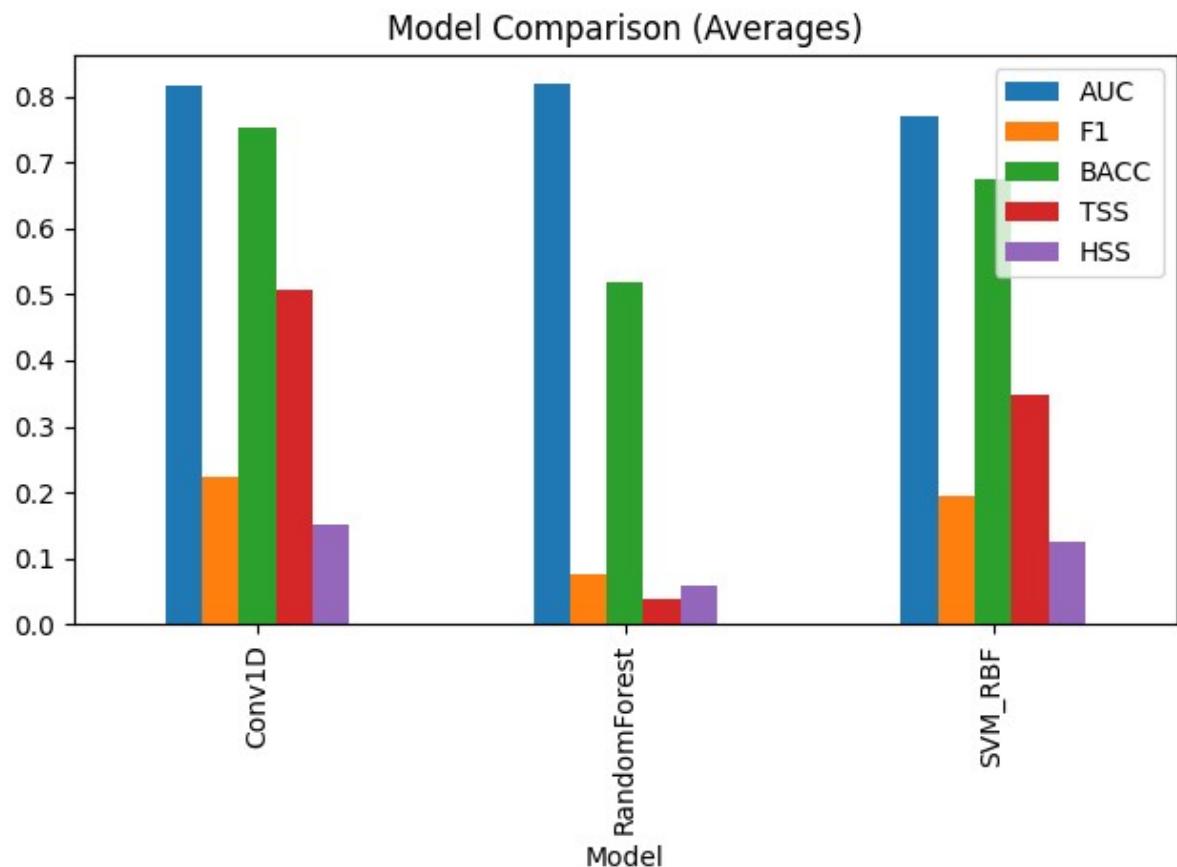
## Conv1D Confusion Matrix



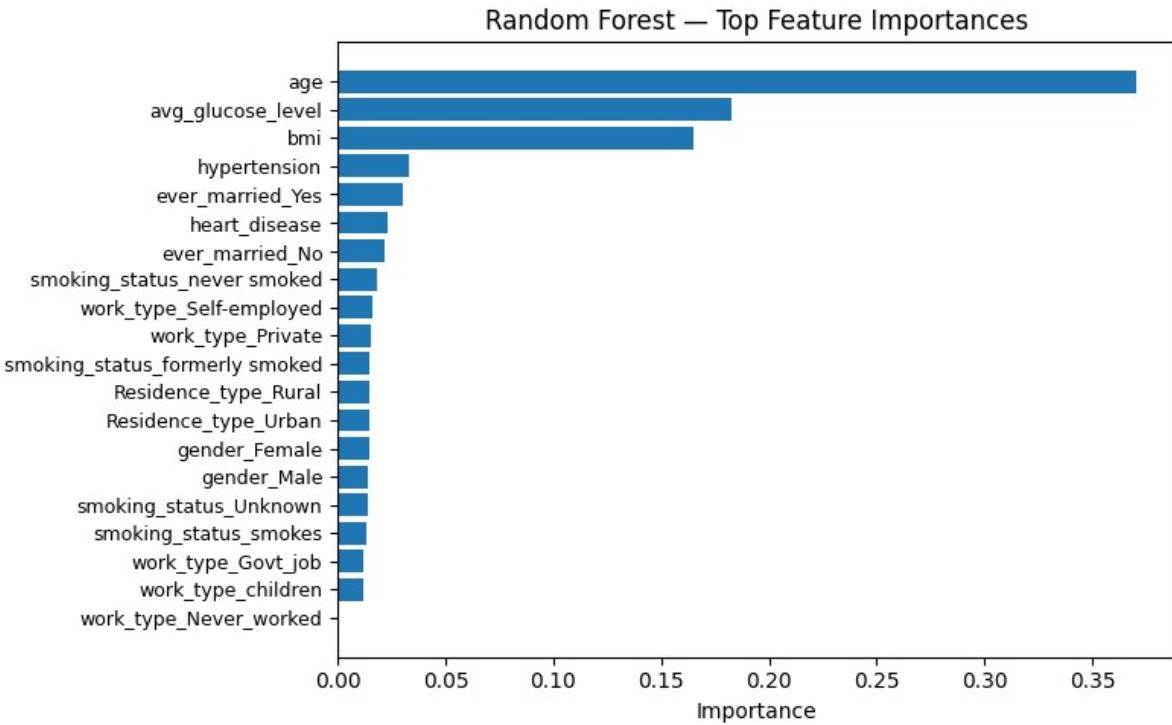
## ROC Curves (All Models)



Model Comparison Bar Chart



Random Forest Feature Importance



## 11. Observations

- All three models clear 0.80 AUC; Random Forest offers the best balance between recall and specificity.
- Conv1D achieves the highest sensitivity, useful when the requirement prioritizes catching every possible stroke case.
- SVM (RBF) sits between the two extremes, demonstrating how kernel methods cope with mixed data once categorical variables are encoded.

**GitHub Repo:** [https://github.com/siddhartha-njit/avadhanam\\_siddharthapreetham\\_finalproject](https://github.com/siddhartha-njit/avadhanam_siddharthapreetham_finalproject)