CS634 Final Project

Name: Pablo Salar Carrera
UCID: ps2255
Instructor: Dr. Yasser
Class: CS634

Tutorial:

Please find below the code and each step to be able to work on it. <br

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sbn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.preprocessing import RobustScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import brier_score_loss
from sklearn.metrics import auc
from sklearn.neighbors import KNeighborsClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Input
from tensorflow.keras.utils import to_categorical
from imblearn.over_sampling import SMOTE
```

```
2024-11-24 23:35:16.741728: I tensorflow/core/platform/cpu_feature_guard.c
c:210] This TensorFlow binary is optimized to use available CPU instruction
s in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuil
d TensorFlow with the appropriate compiler flags.
```

Dataset link:
https://www.kaggle.com/datasets/iamsouravbanerjee/heart-attack-prediction-dataset?resource=download

In [3]:
```
dataset = pd.read_csv('salar_pablo_finaltermproj.csv', index_col = 0)
```

In [4]:
```
dataset.head()
```

Out[4]:

	Age	Sex	Cholesterol	Blood Pressure	Heart Rate	Diabetes	Family History	Smoking	Obesity
Patient ID									
BMW7812	67	Male	208	158/88	72	0	0	1	0
CZE1114	21	Male	389	165/93	98	1	1	1	1
BNI9906	21	Female	324	174/99	72	1	0	0	0
JLN3497	84	Male	383	163/100	73	1	1	1	0
GFO8847	66	Male	318	91/88	93	1	1	1	1

5 rows × 25 columns

In [5]:
```
dataset.shape
```

Out[5]:
```
(8763, 25)
```

In [6]:
```
dataset.dtypes
```

Out[6]:
```
Age                                 int64
Sex                                object
Cholesterol                         int64
Blood Pressure                     object
Heart Rate                          int64
Diabetes                            int64
Family History                      int64
Smoking                             int64
Obesity                             int64
Alcohol Consumption                 int64
Exercise Hours Per Week           float64
Diet                               object
Previous Heart Problems             int64
Medication Use                      int64
Stress Level                        int64
Sedentary Hours Per Day           float64
Income                              int64
BMI                               float64
Triglycerides                       int64
Physical Activity Days Per Week     int64
Sleep Hours Per Day                 int64
Country                            object
Continent                          object
Hemisphere                         object
Heart Attack Risk                   int64
dtype: object
```

Let's check for missing values and duplicates in the dataset.

In [7]:
```
dataset.isna().sum()
```

Out[7]:
```
Age                              0
Sex                              0
Cholesterol                      0
Blood Pressure                   0
Heart Rate                       0
Diabetes                         0
Family History                   0
Smoking                          0
Obesity                          0
Alcohol Consumption              0
Exercise Hours Per Week          0
Diet                             0
Previous Heart Problems          0
Medication Use                   0
Stress Level                     0
Sedentary Hours Per Day          0
Income                           0
BMI                              0
Triglycerides                    0
Physical Activity Days Per Week  0
Sleep Hours Per Day              0
Country                          0
Continent                        0
Hemisphere                       0
Heart Attack Risk                0
dtype: int64
```

In [8]:
```
dataset.duplicated().sum()
```

Out[8]:
```
0
```

There are no missing values or duplicates, therefore we can continue analyzing the data.

Let's see how the numerical columns are distributed

In [9]:
```
dataset.describe()
```

Out[9]:

	Age	Cholesterol	Heart Rate	Diabetes	Family History	Smoking	
count	8763.000000	8763.000000	8763.000000	8763.000000	8763.000000	8763.000000	8
mean	53.707977	259.877211	75.021682	0.652288	0.492982	0.896839	
std	21.249509	80.863276	20.550948	0.476271	0.499979	0.304186	
min	18.000000	120.000000	40.000000	0.000000	0.000000	0.000000	
25%	35.000000	192.000000	57.000000	0.000000	0.000000	1.000000	
50%	54.000000	259.000000	75.000000	1.000000	0.000000	1.000000	
75%	72.000000	330.000000	93.000000	1.000000	1.000000	1.000000	
max	90.000000	400.000000	110.000000	1.000000	1.000000	1.000000	

In [10]:
```
for column in dataset.columns:
    unique_values = dataset[column].unique()
    print(f"Unique values in '{column}': {unique_values}")
    print()
```

```
Unique values in 'Age': [67 21 84 66 54 90 20 43 73 71 77 60 88 69 38 50 45
36 48 40 79 63 27 25
 86 42 52 29 30 47 44 33 51 70 85 31 56 24 74 72 55 26 53 46 57 22 35 39
 80 65 83 82 28 19 75 18 34 37 89 32 49 23 59 62 64 61 76 41 87 81 58 78
 68]

Unique values in 'Sex': ['Male' 'Female']

Unique values in 'Cholesterol': [208 389 324 383 318 297 358 220 145 248 37
3 374 228 259 122 379 166 303
 340 294 359 202 133 159 271 273 328 154 135 197 321 375 360 263 201 347
 129 229 251 121 190 185 279 336 192 180 203 368 222 243 218 120 285 377
 369 311 139 266 153 339 329 333 398 124 183 163 362 390 200 396 255 209
 247 250 227 246 223 330 195 194 178 155 240 237 216 276 224 326 198 301
 314 304 334 213 254 230 316 277 388 206 384 205 261 308 338 382 291 168
 171 378 253 245 226 281 123 173 231 234 268 306 186 293 161 380 239 149
 320 219 335 265 126 307 270 225 193 148 296 136 364 353 252 232 387 299
 357 214 370 345 351 344 152 150 131 272 302 337 170 356 274 188 125 138
 376 181 184 275 394 128 217 399 283 289 284 327 262 212 350 385 162 141
 361 244 295 287 144 354 363 352 140 196 172 319 325 331 392 147 187 346
 286 151 300 165 343 366 317 386 158 157 242 241 365 257 348 175 298 269
 267 397 310 341 204 127 290 280 132 322 179 199 143 312 288 395 189 156
 238 381 391 355 210 400 260 235 167 256 249 207 130 134 137 305 236 315
 292 323 146 258 332 372 142 309 177 367 371 211 282 342 264 176 160 233
 313 164 349 221 191 174 393 278 215 169 182]

Unique values in 'Blood Pressure': ['158/88' '165/93' '174/99' ... '137/94'
'94/76' '119/67']

Unique values in 'Heart Rate': [ 72  98  73  93  48  84 107  68  55  97  70
85 102  40  56 104  71  69
  66  81  52 105  96  74  49  45  50  46  44 106  83  86  65 101  51  43
  79  90  94  78  92  54 109  61  64  82 110  42  63  41 100  76  75  58
  53  60  77  47  59  57  87  67  88  99  80  95 108  89  62 103  91]

Unique values in 'Diabetes': [0 1]

Unique values in 'Family History': [0 1]

Unique values in 'Smoking': [1 0]

Unique values in 'Obesity': [0 1]

Unique values in 'Alcohol Consumption': [0 1]

Unique values in 'Exercise Hours Per Week': [ 4.16818884  1.81324162  2.078
35299 ...  3.14843791  3.78994983
 18.08174797]

Unique values in 'Diet': ['Average' 'Unhealthy' 'Healthy']

Unique values in 'Previous Heart Problems': [0 1]

Unique values in 'Medication Use': [0 1]

Unique values in 'Stress Level': [ 9  1  6  2  7  4  5  8 10  3]

Unique values in 'Sedentary Hours Per Day': [6.61500145 4.96345884 9.463425
84 ... 2.37521373 0.02910426 9.00523438]

Unique values in 'Income': [261404 285768 235282 ...  36998 209943 247338]

Unique values in 'BMI': [31.25123273 27.19497335 28.17657068 ... 35.4061461
6 27.29402009
```

```
 32.91415086]

Unique values in 'Triglycerides': [286 235 587 378 231 795 284 370 790 232
469 523 590 506 635 773  68 402
 517 247 747 360 358 526 605 667 316 551 482 718 297 661 558 209 586 743
 411 785 697 519 595 452 158 679 675 792 584 366 741 474  92 410 398 493
 614 682 106 216 408 628 481  67  82 305 164 211 511 766 547 327 367 681
 131  42 692 664 543 689 569 458 683 779 136 643 653  55 275 314 760 404
 576 690 648 385 255 468 784 509 205 109 530 654 331 485 250 113 377 180
 229 602 285 471 554 344 416 445 709 426 528 388 441 306 749 347 341 451
 356 336 455 223 262 239 555 363 489 788 121 553 617 174 167 563 665  65
 657 237 141 767 292 214 221 447 634 460 711  97 267 695 717 383 332 449
 701 524 549  31 276 744 128  52 394  54 739 407 751 436 473 218 129 579
 492 696 202 197 521 325  35 123 694 434 248 348 750 431 714 649 668 401
 610 244 691  88 532 777 420 350 652 413 754 753 457 122 312 778 676 775
 183 601 317 592 191  83  32 453 423 234 650 565 798 769 412  63 198  93
 764 737  94 298 288 735 190 281 146 574 359 155 719 466 273 515 187 544
 103 132 118 115  85  38 117 362 133 498 645 339 787 733 663 291 502  78
  81 257 624  91 374 270 797 446 464 450 722 556 184 428 796 656 134 196
 623 522 376 730 463  99 593  47 148 302  57 280 389 629 294 186 700 774
 181 375 467 603 616 380 495 698 318 207 780  51  84 425 310 126  56 472
 669 688 655  39 333 501 479 540 433 179 490 204 644 525 546 486 320 319
  58 591 165 732 195 478 461 631 301  50 315 194 199 160 149 527 406 161
 125 200 277 308  69 427 236  77 500 269  79 168 575 606 355 636  64 251
 245 228 287 800 483 791 260 604 536 559 124 254 159  73 542 390 755  60
  61 491  40 437 215 440 379 789 266 505 243 783 403 637 156 729 438 507
 725 562 324  87 253 626 541 364 456  30 182 621 494 776 442 429 684 219
  70  98 166  95 135 646 337 226 710 608 208 724 704 512 206 224 622 598
 465 119 293 630 386 513  45 578 261 217 715 282 391 580 192 399 249 396
 278 448 782 419 503 220  49 304 157 150 545 627 582 178 263  33 299 303
  66 763 256 139 651 756 372 345  48  46 421  43 771 210 781  41 508 353
 566 726 736 326 759 477 369 188 104 329 309 384 599 415 770 571 552 145
 632 373  71 550 583 322 475 357 673 454 757 201 100 274 258 613 233 330
 731 761 296 573 335 716 642 142 674 572 638 222 752 740 397 594 705 381
 615 539 242 499 435 680 535 238 283  89 589 666 678  76 176 620  75 721
 143 723 570  44 203 259 677 734 662 707 745 487 577 443 120 111 365 116
 538 162 742 212 581 313  36 400 619 609 252 706 264 290 138 300 346 712
  34 387 140 154 758 462 672 713  86 414 699 529 382 432 368 193  72 537
 560 189 342 531 311 241 685 497 640 321 480 144 585 171 727 660 799 600
 597 213 708 151 265 618 658 746 307  53 514 611 153 352 225 567 702 520
 417 102 607 548 647 476 762 147 424 459 409  74 510  37 323 240 175 786
  80 439 504 772 670  59 334 703 392  90 496 422 279 343 671 794 163 328
 625 272 227 152 105 693  96 484 568 633 659 230 112 793 101 172 110 612
 185 289 418 533 686 641 169 349 173 516  62 557 596 728 371 738 444 561
 114 765 338 588 246 295 564 488 177 687 395 518 127 639 137 354 271 107
 340 534 768 130 720 405 430 268 108 748 351 393 361 170 470]

Unique values in 'Physical Activity Days Per Week': [0 1 4 3 5 6 7 2]

Unique values in 'Sleep Hours Per Day': [ 6  7  4  5 10  8  9]

Unique values in 'Country': ['Argentina' 'Canada' 'France' 'Thailand' 'Germ
any' 'Japan' 'Brazil'
 'South Africa' 'United States' 'Vietnam' 'China' 'Italy' 'Spain' 'India'
 'Nigeria' 'New Zealand' 'South Korea' 'Australia' 'Colombia'
 'United Kingdom']

Unique values in 'Continent': ['South America' 'North America' 'Europe' 'As
ia' 'Africa' 'Australia']

Unique values in 'Hemisphere': ['Southern Hemisphere' 'Northern Hemispher
e']
```

```
Unique values in 'Heart Attack Risk': [0 1]
```

In [11]:

```
for column in dataset.columns:
    print(f"Value counts for '{column}':")
    print(dataset[column].value_counts())
    print()
```

```
Value counts for 'Age':
Age
90    152
42    150
33    147
59    147
29    137
     ...
75    102
72    101
39    100
47     99
51     82
Name: count, Length: 73, dtype: int64

Value counts for 'Sex':
Sex
Male      6111
Female    2652
Name: count, dtype: int64

Value counts for 'Cholesterol':
Cholesterol
235    52
360    47
149    46
218    46
251    45
       ..
248    20
186    20
328    20
398    20
397    19
Name: count, Length: 281, dtype: int64

Value counts for 'Blood Pressure':
Blood Pressure
146/94     8
101/93     8
106/64     7
102/104    7
176/77     7
          ..
155/102    1
154/71     1
178/90     1
98/85      1
119/67     1
Name: count, Length: 3915, dtype: int64

Value counts for 'Heart Rate':
Heart Rate
94     157
97     146
57     143
52     140
104    139
      ...
70     107
48     107
79     105
96      97
73      93
```

```
Name: count, Length: 71, dtype: int64

Value counts for 'Diabetes':
Diabetes
1    5716
0    3047
Name: count, dtype: int64

Value counts for 'Family History':
Family History
0    4443
1    4320
Name: count, dtype: int64

Value counts for 'Smoking':
Smoking
1    7859
0     904
Name: count, dtype: int64

Value counts for 'Obesity':
Obesity
1    4394
0    4369
Name: count, dtype: int64

Value counts for 'Alcohol Consumption':
Alcohol Consumption
1    5241
0    3522
Name: count, dtype: int64

Value counts for 'Exercise Hours Per Week':
Exercise Hours Per Week
4.168189     1
18.477430    1
11.883523    1
19.353157    1
19.365546    1
            ..
9.884039     1
12.644947    1
1.089868     1
10.500477    1
18.081748    1
Name: count, Length: 8763, dtype: int64

Value counts for 'Diet':
Diet
Healthy      2960
Average      2912
Unhealthy    2891
Name: count, dtype: int64

Value counts for 'Previous Heart Problems':
Previous Heart Problems
0    4418
1    4345
Name: count, dtype: int64

Value counts for 'Medication Use':
Medication Use
0    4396
1    4367
```

```
Name: count, dtype: int64

Value counts for 'Stress Level':
Stress Level
2     913
4     910
7     903
9     887
8     879
3     868
1     865
5     860
6     855
10    823
Name: count, dtype: int64

Value counts for 'Sedentary Hours Per Day':
Sedentary Hours Per Day
6.615001     1
0.772688     1
0.723868     1
10.125510    1
2.054331     1
            ..
11.921800    1
0.087028     1
9.198925     1
3.383760     1
9.005234     1
Name: count, Length: 8763, dtype: int64

Value counts for 'Income':
Income
225278    4
194461    3
195282    3
220507    2
139451    2
         ..
44744     1
85563     1
20443     1
258704    1
247338    1
Name: count, Length: 8615, dtype: int64

Value counts for 'BMI':
BMI
31.251233    1
39.385227    1
36.280438    1
18.218558    1
23.885840    1
            ..
28.358868    1
22.539845    1
34.721372    1
18.881817    1
32.914151    1
Name: count, Length: 8763, dtype: int64

Value counts for 'Triglycerides':
Triglycerides
799    25
```

```
507    22
121    22
593    22
469    22
       ..
120     3
213     3
185     3
295     3
130     2
Name: count, Length: 771, dtype: int64

Value counts for 'Physical Activity Days Per Week':
Physical Activity Days Per Week
3    1143
1    1121
2    1109
7    1095
5    1079
4    1077
6    1074
0    1065
Name: count, dtype: int64

Value counts for 'Sleep Hours Per Day':
Sleep Hours Per Day
10    1293
8     1288
6     1276
7     1270
5     1263
9     1192
4     1181
Name: count, dtype: int64

Value counts for 'Country':
Country
Germany           477
Argentina         471
Brazil            462
United Kingdom    457
Australia         449
Nigeria           448
France            446
Canada            440
China             436
New Zealand       435
Japan             433
Italy             431
Spain             430
Colombia          429
Thailand          428
South Africa      425
Vietnam           425
United States     420
India             412
South Korea       409
Name: count, dtype: int64

Value counts for 'Continent':
Continent
Asia             2543
Europe           2241
South America    1362
```

```
Australia        884
Africa           873
North America    860
Name: count, dtype: int64

Value counts for 'Hemisphere':
Hemisphere
Northern Hemisphere    5660
Southern Hemisphere    3103
Name: count, dtype: int64

Value counts for 'Heart Attack Risk':
Heart Attack Risk
0    5624
1    3139
Name: count, dtype: int64
```

We group the numerical and categorical columns for future preprocessing and analysis

In [12]:
```
numerical_columns = dataset.select_dtypes(include=['int64', 'float64']).colu
categorical_columns = dataset.select_dtypes(include=['object']).columns.toli

print(len(numerical_columns), "Numerical Columns: ", numerical_columns)
print(len(categorical_columns), "Categorical Columns: ", categorical_column
```

```
19 Numerical Columns:  ['Age', 'Cholesterol', 'Heart Rate', 'Diabetes', 'Fa
mily History', 'Smoking', 'Obesity', 'Alcohol Consumption', 'Exercise Hours
Per Week', 'Previous Heart Problems', 'Medication Use', 'Stress Level', 'Se
dentary Hours Per Day', 'Income', 'BMI', 'Triglycerides', 'Physical Activit
y Days Per Week', 'Sleep Hours Per Day', 'Heart Attack Risk']
6 Categorical Columns:  ['Sex', 'Blood Pressure', 'Diet', 'Country', 'Conti
nent', 'Hemisphere']
```

In [13]:
```
for col in numerical_columns:
    print(f"\nDistribution of 'Heart Attack Risk' by {col}:")
    print(pd.crosstab(dataset[col], dataset["Heart Attack Risk"]))
```

```
Distribution of 'Heart Attack Risk' by Age:
```

Heart Attack Risk	0	1
Age		
18	82	41
19	88	40
20	91	39
21	76	41
22	84	40
...	...	...
86	59	46
87	72	54
88	82	41
89	74	43
90	97	55

73 rows × 2 columns

```
Distribution of 'Heart Attack Risk' by Cholesterol:
```

Heart Attack Risk	0	1
Cholesterol		
120	22	10
121	25	8
122	17	14
123	21	10
124	26	8
...	...	...
396	22	10
397	11	8
398	13	7
399	29	5
400	22	12

281 rows × 2 columns

```
Distribution of 'Heart Attack Risk' by Heart Rate:
```

Heart Attack Risk	0	1
Heart Rate		
40	77	37
41	85	51
42	81	48
43	75	36
44	85	45
...	...	...
106	74	37
107	75	43
108	72	50
109	83	47
110	78	48

71 rows × 2 columns

```
Distribution of 'Heart Attack Risk' by Diabetes:
```

Heart Attack Risk	0	1
Diabetes		
0	1990	1057
1	3634	2082

```
Distribution of 'Heart Attack Risk' by Family History:
```

Heart Attack Risk	0	1
Family History		
0	2848	1595
1	2776	1544

```
Distribution of 'Heart Attack Risk' by Smoking:
```

Heart Attack Risk	0	1
Smoking		
0	575	329
1	5049	2810

```
Distribution of 'Heart Attack Risk' by Obesity:
```

Heart Attack Risk	0	1
Obesity		
0	2776	1593
1	2848	1546

```
Distribution of 'Heart Attack Risk' by Alcohol Consumption:
```

Heart Attack Risk	0	1
Alcohol Consumption		
0	2232	1290
1	3392	1849

```
Distribution of 'Heart Attack Risk' by Exercise Hours Per Week:
```

Heart Attack Risk	0	1
Exercise Hours Per Week		
0.002442	1	0
0.004443	1	0
0.005109	0	1
0.007422	0	1
0.008115	0	1
...	...	...
19.986355	1	0
19.990822	0	1
19.997012	0	1
19.997891	1	0
19.998709	1	0

8763 rows × 2 columns

```
Distribution of 'Heart Attack Risk' by Previous Heart Problems:
```

Heart Attack Risk	0	1
Previous Heart Problems		
0	2836	1582
1	2788	1557

```
Distribution of 'Heart Attack Risk' by Medication Use:
```

Heart Attack Risk	0	1
Medication Use		
0	2826	1570
1	2798	1569

```
Distribution of 'Heart Attack Risk' by Stress Level:
```

Heart Attack Risk	0	1
Stress Level		
1	562	303
2	583	330
3	551	317
4	597	313
5	543	317
6	533	322
7	567	336
8	568	311
9	584	303
10	536	287

```
Distribution of 'Heart Attack Risk' by Sedentary Hours Per Day:
```

Heart Attack Risk	0	1
Sedentary Hours Per Day		
0.001263	1	0
0.001529	0	1
0.003625	1	0
0.008307	1	0
0.010631	1	0
...	...	...
11.985484	0	1
11.987716	1	0
11.989217	0	1
11.992341	1	0
11.999313	0	1

8763 rows × 2 columns

```
Distribution of 'Heart Attack Risk' by Income:
```

Heart Attack Risk	0	1
Income		
20062	1	0
20140	1	0
20162	1	0
20165	1	0
20208	1	0
...	...	...
299810	1	0
299850	0	1
299891	1	0
299909	0	1
299954	0	1

8615 rows × 2 columns

```
Distribution of 'Heart Attack Risk' by BMI:
```

Heart Attack Risk	0	1
BMI		
18.002337	1	0
18.004211	1	0
18.009025	1	0
18.013606	1	0
18.016191	1	0
...	...	...
39.985120	0	1
39.986127	0	1
39.989915	1	0
39.993581	1	0
39.997211	0	1

8763 rows × 2 columns

```
Distribution of 'Heart Attack Risk' by Triglycerides:
```

Heart Attack Risk	0	1
Triglycerides		
30	12	4
31	6	8
32	8	3
33	8	5
34	5	1
...	...	...
796	7	6
797	16	3
798	8	4
799	19	6
800	5	3

771 rows × 2 columns

```
Distribution of 'Heart Attack Risk' by Physical Activity Days Per Week:
```

Heart Attack Risk	0	1
Physical Activity Days Per Week		
0	651	414
1	733	388
2	716	393
3	742	401
4	692	385
5	710	369
6	695	379
7	685	410

```
Distribution of 'Heart Attack Risk' by Sleep Hours Per Day:
```

Heart Attack Risk	0	1
Sleep Hours Per Day		
4	752	429
5	801	462
6	807	469
7	816	454
8	807	481
9	778	414
10	863	430

```
Distribution of 'Heart Attack Risk' by Heart Attack Risk:
```

Heart Attack Risk	0	1
Heart Attack Risk		
0	5624	0
1	0	3139

In [14]:
```
for col in categorical_columns:
    print(f"\nDistribution of 'Heart Attack Risk' by {col}:")
    print(pd.crosstab(dataset[col], dataset["Heart Attack Risk"]))
```

```
Distribution of 'Heart Attack Risk' by Sex:
```

Heart Attack Risk	0	1
Sex		
Female	1708	944
Male	3916	2195

```
Distribution of 'Heart Attack Risk' by Blood Pressure:
```

Heart Attack Risk	0	1
Blood Pressure		
100/100	2	0
100/102	1	0
100/103	3	1
100/104	4	0
100/105	3	1
...	...	...
99/92	2	1
99/93	1	0
99/94	2	0
99/96	3	2
99/98	1	1

3915 rows × 2 columns

```
Distribution of 'Heart Attack Risk' by Diet:
```

Heart Attack Risk	0	1
Diet		
Average	1886	1026
Healthy	1881	1079
Unhealthy	1857	1034

```
Distribution of 'Heart Attack Risk' by Country:
```

Heart Attack Risk	0	1
Country		
Argentina	297	174
Australia	281	168
Brazil	299	163
Canada	282	158
China	281	155
Colombia	267	162
France	289	157
Germany	305	172
India	283	129
Italy	295	136
Japan	289	144
New Zealand	284	151
Nigeria	270	178
South Africa	281	144
South Korea	246	163
Spain	280	150
Thailand	267	161
United Kingdom	297	160
United States	254	166
Vietnam	277	148

```
Distribution of 'Heart Attack Risk' by Continent:
```

Heart Attack Risk	0	1
Continent		
Africa	551	322
Asia	1643	900
Australia	565	319
Europe	1466	775
North America	536	324
South America	863	499

```
Distribution of 'Heart Attack Risk' by Hemisphere:
```

Heart Attack Risk	0	1
Hemisphere		
Northern Hemisphere	3607	2053
Southern Hemisphere	2017	1086

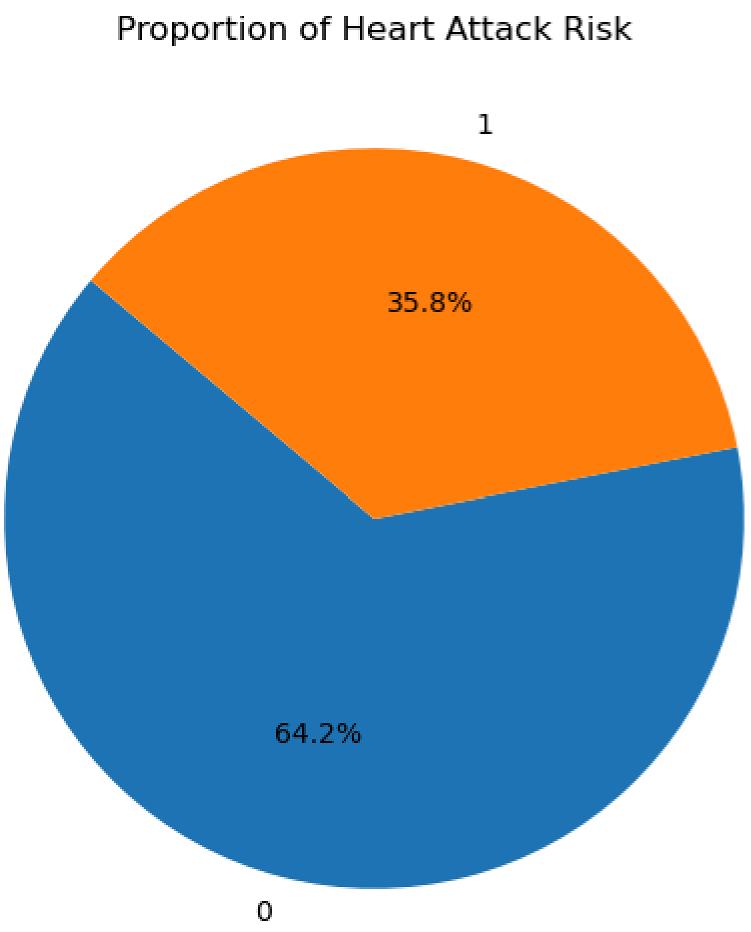
In [15]:
```
target_column = 'Heart Attack Risk'
```

In [16]:
```
target_counts = dataset[target_column].value_counts()
print("Counts of each target category:\n", target_counts)
```

```
Counts of each target category:
 Heart Attack Risk
0    5624
1    3139
Name: count, dtype: int64
```

We check how the target data is distributed. There are 5624 cases in which there is no presence of heart attack risk, while 3139 cases where there exists a risk.

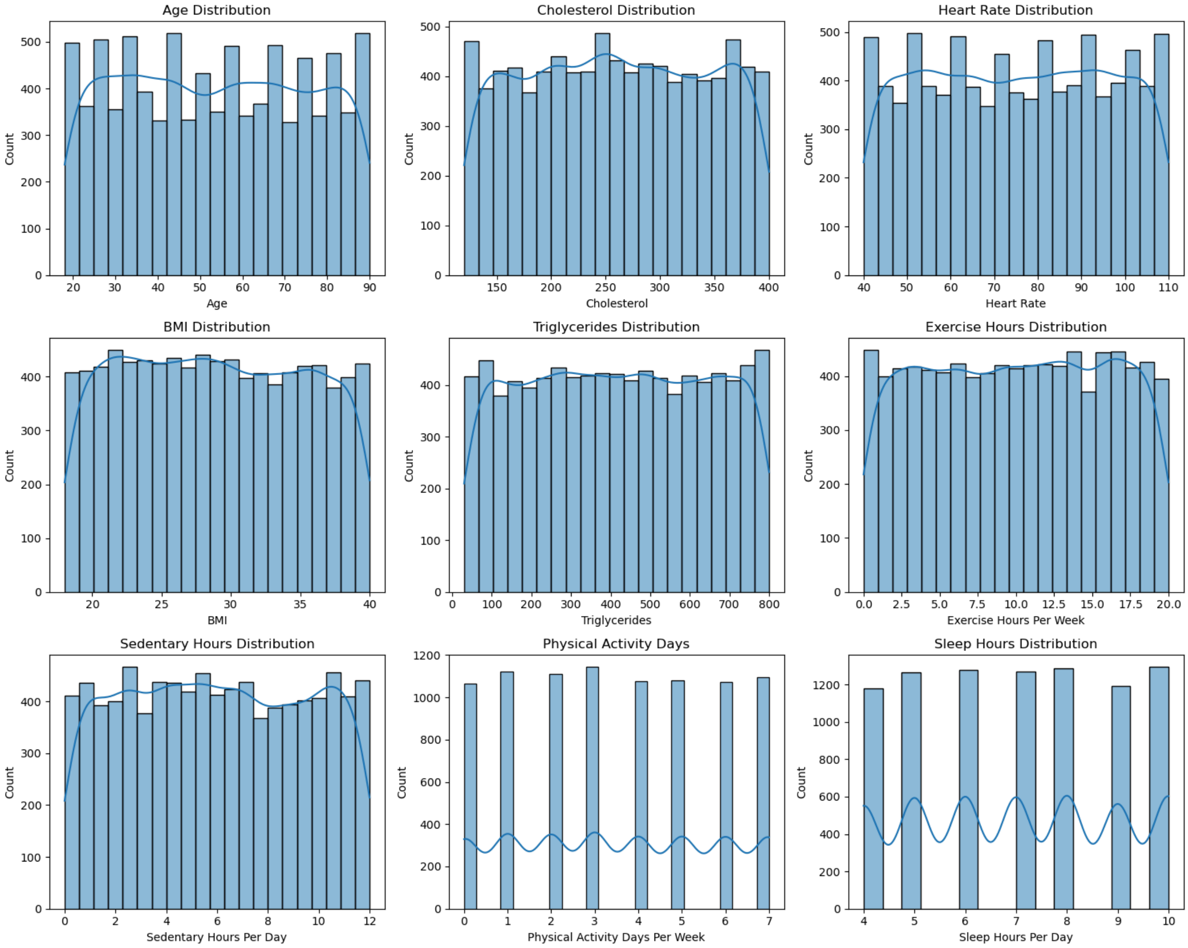
In [17]:
```
plt.figure(figsize=(6, 6))
dataset[target_column].value_counts().plot.pie(autopct='%1.1f%%', startangle
plt.title(f"Proportion of {target_column}")
plt.ylabel("")  # Hide the y-label
plt.show()
```



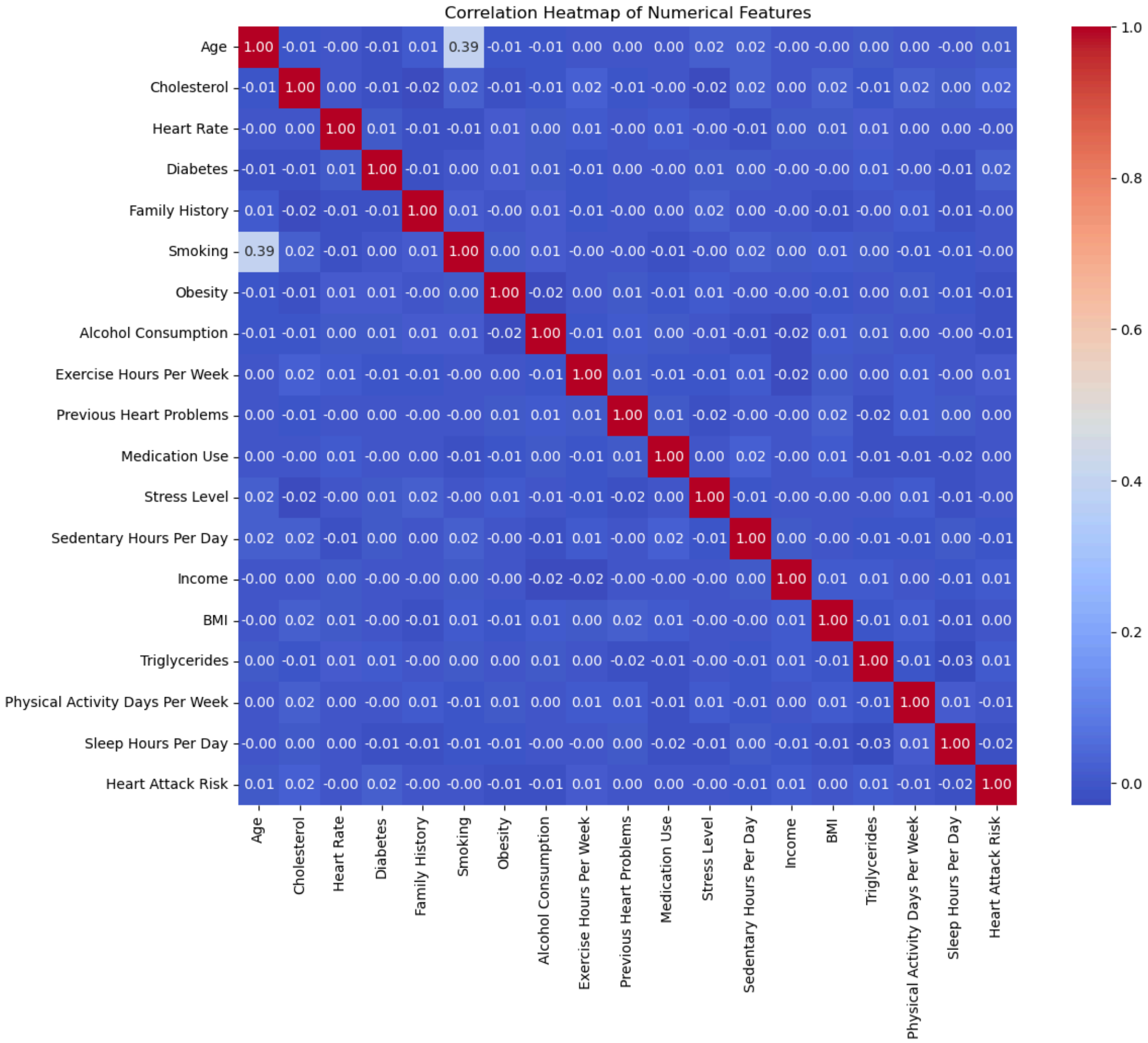
The proportion of the target variable, Heart Attack Risk, is 35.8% of presence of heart attack risk and 64.2% of no presence of heart attack risk.

In [18]:
```
# Plotting histograms for a few key numerical features
fig, axs = plt.subplots(3, 3, figsize=(15, 12))
sbn.histplot(dataset['Age'], kde=True, ax=axs[0, 0]).set(title="Age Distribu
sbn.histplot(dataset['Cholesterol'], kde=True, ax=axs[0, 1]).set(title="Cho
sbn.histplot(dataset['Heart Rate'], kde=True, ax=axs[0, 2]).set(title="Hear
sbn.histplot(dataset['BMI'], kde=True, ax=axs[1, 0]).set(title="BMI Distribu
sbn.histplot(dataset['Triglycerides'], kde=True, ax=axs[1, 1]).set(title="Tr
sbn.histplot(dataset['Exercise Hours Per Week'], kde=True, ax=axs[1, 2]).se
sbn.histplot(dataset['Sedentary Hours Per Day'], kde=True, ax=axs[2, 0]).se
```

```
sbn.histplot(dataset['Physical Activity Days Per Week'], kde=True, ax=axs[2,
sbn.histplot(dataset['Sleep Hours Per Day'], kde=True, ax=axs[2, 2]).set(ti
plt.tight_layout()
plt.show()
```

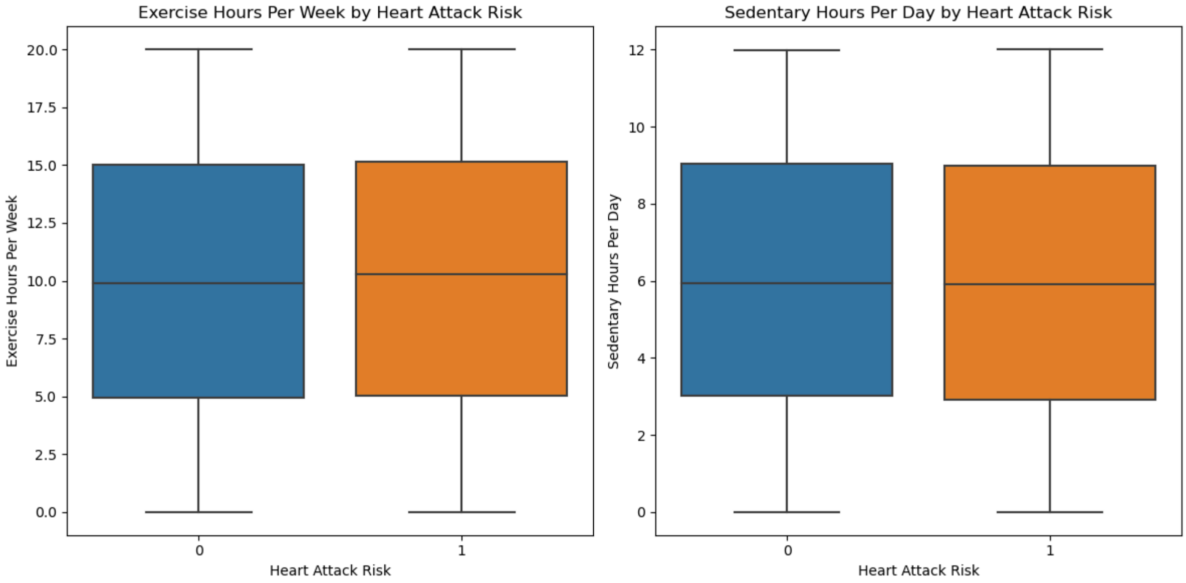


In [19]:
```
# Correlation heatmap
numerical_data = dataset.select_dtypes(include=['int64', 'float64'])
plt.figure(figsize=(14, 10))
sbn.heatmap(numerical_data.corr(), annot=True, fmt=".2f", cmap="coolwarm", s
plt.title("Correlation Heatmap of Numerical Features")
plt.show()
```



We check for any correlations between the numerical data, but we do not find any highly correlated features. There is only a slightly positive correlation between the smoking and the age of the patients of the dataset.

In [20]:
```
# Plotting Exercise Hours and Sedentary Hours against Heart Attack Risk
fig, axs = plt.subplots(1, 2, figsize=(12, 6))

sbn.boxplot(x='Heart Attack Risk', y='Exercise Hours Per Week', data=datase
axs[0].set_title('Exercise Hours Per Week by Heart Attack Risk')

sbn.boxplot(x='Heart Attack Risk', y='Sedentary Hours Per Day', data=datase
axs[1].set_title('Sedentary Hours Per Day by Heart Attack Risk')

plt.tight_layout()
plt.show()
```



Define the function to calculate the metrics using the formulas from the slides.

In [21]:
```
def calculate_metrics(y_true, y_pred):
    """
    Calculate evaluation metrics based on the confusion matrix.

    Parameters:
        y_true (list or array): True labels.
        y_pred (list or array): Predicted labels.

    Returns:
        pd.DataFrame: DataFrame containing the metrics.
    """
    # Get confusion matrix values
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()

    # Calculate metrics manually
    fpr = fp / (fp + tn) if (fp + tn) != 0 else 0
    fnr = fn / (fn + tp) if (fn + tp) != 0 else 0
    tss = (tp / (tp + fn) if (tp + fn) != 0 else 0) - fpr
    hss = (2 * (tp * tn - fp * fn)) / (
        ((tp + fn) * (fn + tn) + (tp + fp) * (fp + tn))
    ) if ((tp + fn) * (fn + tn) + (tp + fp) * (fp + tn)) != 0 else 0
    accuracy = (tp + tn) / (tp + tn + fp + fn)
    precision = tp / (tp + fp) if (tp + fp) != 0 else 0
    recall = tp / (tp + fn) if (tp + fn) != 0 else 0
    specificity = tn / (tn + fp) if (tn + fp) != 0 else 0
    f1 = (2 * precision * recall) / (precision + recall) if (precision + rec
    error_rate = 1 - accuracy
    balanced_acc = (recall + specificity) / 2


    # Store the metrics
    metrics = {
        "FPR" : fpr,
        "FNR" : fnr,
        "TSS" : tss,
        "HSS" : hss,
        "Accuracy" : accuracy,
        "Precision" : precision,
        "Recall/Sensitivity" : recall,
        "Specificity" : specificity,
        "F1 Measure" : f1,
        "Error Rate" : error_rate,
```

```
        "Balanced Accuracy" : balanced_acc
    }
    return metrics
```

We use LabelEncoder to convert the object variables into numbers so our models can use the data.

In [22]:
```
# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Apply Label Encoding to each categorical column
for col in categorical_columns:
    dataset[col] = label_encoder.fit_transform(dataset[col])
```

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

Out[23]:

Patient ID	Age	Sex	Cholesterol	Blood Pressure	Heart Rate	Diabetes	Family History	Smoking	Obesity	C
BMW7812	67	1	208	2510	72	0	0	1	0	
CZE1114	21	1	389	2815	98	1	1	1	1	
BNI9906	21	0	324	3224	72	1	0	0	0	
JLN3497	84	1	383	2689	73	1	1	1	0	
GFO8847	66	1	318	3563	93	1	1	1	1	

5 rows × 25 columns

For a better use of the models, we standarize the numerical features using RobustScaler()

In [24]:
```
# Standardize numerical features
scaler = RobustScaler()
dataset[numerical_columns] = scaler.fit_transform(dataset[numerical_columns]
```

In [25]:
```
dataset.head()
```

Out[25]:

Patient ID	Age	Sex	Cholesterol	Blood Pressure	Heart Rate	Diabetes	Family History	Smoking	C
BMW7812	0.351351	1	-0.369565	2510	-0.083333	-1.0	0.0	0.0	
CZE1114	-0.891892	1	0.942029	2815	0.638889	0.0	1.0	0.0	
BNI9906	-0.891892	0	0.471014	3224	-0.083333	0.0	0.0	-1.0	
JLN3497	0.810811	1	0.898551	2689	-0.055556	0.0	1.0	0.0	
GFO8847	0.324324	1	0.427536	3563	0.500000	0.0	1.0	0.0	

5 rows × 25 columns

In [26]:
```
# Splitting features and target
X = dataset.drop(columns=['Heart Attack Risk'])
y = dataset['Heart Attack Risk']
```

In [27]:
```
# Define the 10-fold cross-validator
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

In [28]:
```
randomforest = RandomForestClassifier(random_state=42)
KNN = KNeighborsClassifier()
```

In [29]:
```
# Define parameter grids for hyperparameter tuning
KNN_param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}

randomforest_param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30]
}
```

In [31]:
```
# Perform grid search for KNN
KNN_grid_search = GridSearchCV(KNN, KNN_param_grid, cv=5, verbose=1, n_jobs=
KNN_grid_search.fit(X, y)
best_KNN = KNN_grid_search.best_estimator_
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

In [32]:
```
best_KNN
```

Out[32]:

```
KNeighborsClassifier

KNeighborsClassifier(metric='euclidean', n_neighbors=9)
```

In [33]:
```
# Perform grid search for Random Forest
randomforest_grid_search = GridSearchCV(randomforest, randomforest_param_gr
randomforest_grid_search.fit(X, y)
best_randomforest = randomforest_grid_search.best_estimator_
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

In [34]:
```
best_randomforest
```

Out[34]:

```
RandomForestClassifier

RandomForestClassifier(max_depth=10, n_estimators=200, random_state
=42)
```

In [35]:
```
all_metrics = []
```

In [36]:
```
# Loop through each fold
for fold, (train_index, test_index) in enumerate(kf.split(X, y)):
    print(f"----- Metrics for all Algorithms in Iteration {fold + 1} -----")

    # Split data
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # ---- Random Forest ----
```

```
    best_randomforest.fit(X_train, y_train)
    y_pred_randomforest = best_randomforest.predict(X_test)
    randomforest_metrics = calculate_metrics(y_test, y_pred_randomforest)

    # ---- KNN ----
    best_KNN.fit(X_train, y_train)
    y_pred_KNN = best_KNN.predict(X_test)
    KNN_metrics = calculate_metrics(y_test, y_pred_KNN)

    # ---- LSTM ----
    # Reshape for LSTM input (samples, timesteps, features)
    X_train_lstm = X_train.values.reshape(X_train.shape[0], 1, X_train.shape
    X_test_lstm = X_test.values.reshape(X_test.shape[0], 1, X_test.shape[1]

    # Convert target to categorical for LSTM
    y_train_lstm = to_categorical(y_train, num_classes=2)
    y_test_lstm = to_categorical(y_test, num_classes=2)

    # Define LSTM model
    lstm_model = Sequential([
        Input(shape=(1, X_train.shape[1])),
        LSTM(64, activation='relu'),
        Dropout(0.2),
        Dense(2, activation='softmax')
    ])
    lstm_model.compile(optimizer='adam', loss='categorical_crossentropy', me
    lstm_model.fit(X_train_lstm, y_train_lstm, epochs=10, batch_size=16, ve

    y_pred_lstm = np.argmax(lstm_model.predict(X_test_lstm), axis=1)

    # Compute LSTM metrics
    lstm_metrics = calculate_metrics(y_test, y_pred_lstm)

    # ---- Combine Metrics ----
    all_metrics_fold = {
        'KNN': KNN_metrics,
        'Random Forest': randomforest_metrics,
        'LSTM': lstm_metrics
    }

    # Store metrics for the current fold
    all_metrics.append(all_metrics_fold)

    # ---- Print Metrics for All Algorithms in Current Iteration ----
    iteration_metrics = pd.DataFrame({
        'Metric': list(KNN_metrics.keys()),
        'KNN': list(KNN_metrics.values()),
        'Random Forest': list(randomforest_metrics.values()),
        'LSTM': list(lstm_metrics.values())
    })
    print(iteration_metrics.to_string(index=False))
```

```
----- Metrics for all Algorithms in Iteration 1 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
              Metric       KNN  Random Forest      LSTM
                 FPR  0.165187       0.000000  0.000000
                 FNR  0.859873       1.000000  0.993631
                 TSS -0.025059       0.000000  0.006369
                 HSS -0.028628       0.000000  0.008163
            Accuracy  0.586089       0.641961  0.644242
           Precision  0.321168       0.000000  1.000000
  Recall/Sensitivity  0.140127       0.000000  0.006369
         Specificity  0.834813       1.000000  1.000000
          F1 Measure  0.195122       0.000000  0.012658
          Error Rate  0.413911       0.358039  0.355758
   Balanced Accuracy  0.487470       0.500000  0.503185
----- Metrics for all Algorithms in Iteration 2 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step
              Metric       KNN  Random Forest      LSTM
                 FPR  0.211368       0.000000  0.001776
                 FNR  0.815287       1.000000  0.993631
                 TSS -0.026654       0.000000  0.004593
                 HSS -0.029501       0.000000  0.005881
            Accuracy  0.572406       0.641961  0.643101
           Precision  0.327684       0.000000  0.666667
  Recall/Sensitivity  0.184713       0.000000  0.006369
         Specificity  0.788632       1.000000  0.998224
          F1 Measure  0.236253       0.000000  0.012618
          Error Rate  0.427594       0.358039  0.356899
   Balanced Accuracy  0.486673       0.500000  0.502297
----- Metrics for all Algorithms in Iteration 3 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
              Metric       KNN  Random Forest      LSTM
                 FPR  0.190053       0.000000  0.000000
                 FNR  0.875796       1.000000  1.000000
                 TSS -0.065849       0.000000  0.000000
                 HSS -0.074686       0.000000  0.000000
            Accuracy  0.564424       0.641961  0.641961
           Precision  0.267123       0.000000  0.000000
  Recall/Sensitivity  0.124204       0.000000  0.000000
         Specificity  0.809947       1.000000  1.000000
          F1 Measure  0.169565       0.000000  0.000000
          Error Rate  0.435576       0.358039  0.358039
   Balanced Accuracy  0.467075       0.500000  0.500000
----- Metrics for all Algorithms in Iteration 4 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
              Metric      KNN  Random Forest      LSTM
                 FPR 0.170515       0.000000  0.000000
                 FNR 0.814696       1.000000  0.996805
                 TSS 0.014788       0.000000  0.003195
                 HSS 0.016668       0.000000  0.004103
            Accuracy 0.599315       0.642694  0.643836
           Precision 0.376623       0.000000  1.000000
  Recall/Sensitivity 0.185304       0.000000  0.003195
         Specificity 0.829485       1.000000  1.000000
          F1 Measure 0.248394       0.000000  0.006369
          Error Rate 0.400685       0.357306  0.356164
   Balanced Accuracy 0.507394       0.500000  0.501597
----- Metrics for all Algorithms in Iteration 5 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
              Metric      KNN  Random Forest       LSTM
                 FPR 0.192171       0.000000   0.008897
                 FNR 0.777070       1.000000   1.000000
                 TSS 0.030759       0.000000  -0.008897
                 HSS 0.034009       0.000000  -0.011364
            Accuracy 0.598174       0.641553   0.635845
```

```
          Precision 0.393258       0.000000  0.000000
Recall/Sensitivity 0.222930       0.000000  0.000000
        Specificity 0.807829       1.000000  0.991103
         F1 Measure 0.284553       0.000000  0.000000
         Error Rate 0.401826       0.358447  0.364155
  Balanced Accuracy 0.515380       0.500000  0.495552
----- Metrics for all Algorithms in Iteration 6 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
             Metric        KNN  Random Forest     LSTM
                FPR  0.202847       0.000000 0.000000
                FNR  0.808917       1.000000 0.996815
                TSS -0.011764       0.000000 0.003185
                HSS -0.013048       0.000000 0.004083
           Accuracy  0.579909       0.641553 0.642694
          Precision  0.344828       0.000000 1.000000
Recall/Sensitivity  0.191083       0.000000 0.003185
        Specificity  0.797153       1.000000 1.000000
         F1 Measure  0.245902       0.000000 0.006349
         Error Rate  0.420091       0.358447 0.357306
  Balanced Accuracy  0.494118       0.500000 0.501592
----- Metrics for all Algorithms in Iteration 7 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
             Metric        KNN  Random Forest     LSTM
                FPR  0.202847       0.000000 0.000000
                FNR  0.805732       1.000000 1.000000
                TSS -0.008579       0.000000 0.000000
                HSS -0.009508       0.000000 0.000000
           Accuracy  0.581050       0.641553 0.641553
          Precision  0.348571       0.000000 0.000000
Recall/Sensitivity  0.194268       0.000000 0.000000
        Specificity  0.797153       1.000000 1.000000
         F1 Measure  0.249489       0.000000 0.000000
         Error Rate  0.418950       0.358447 0.358447
  Balanced Accuracy  0.495710       0.500000 0.500000
----- Metrics for all Algorithms in Iteration 8 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step
             Metric        KNN  Random Forest      LSTM
                FPR  0.158363       0.000000  0.007117
                FNR  0.853503       1.000000  1.000000
                TSS -0.011866       0.000000 -0.007117
                HSS -0.013573       0.000000 -0.009100
           Accuracy  0.592466       0.641553  0.636986
          Precision  0.340741       0.000000  0.000000
Recall/Sensitivity  0.146497       0.000000  0.000000
        Specificity  0.841637       1.000000  0.992883
         F1 Measure  0.204900       0.000000  0.000000
         Error Rate  0.407534       0.358447  0.363014
  Balanced Accuracy  0.494067       0.500000  0.496441
----- Metrics for all Algorithms in Iteration 9 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
             Metric       KNN  Random Forest     LSTM
                FPR 0.170819       0.000000 0.000000
                FNR 0.805732       1.000000 1.000000
                TSS 0.023449       0.000000 0.000000
                HSS 0.026357       0.000000 0.000000
           Accuracy 0.601598       0.641553 0.641553
          Precision 0.388535       0.000000 0.000000
Recall/Sensitivity 0.194268       0.000000 0.000000
        Specificity 0.829181       1.000000 1.000000
         F1 Measure 0.259023       0.000000 0.000000
         Error Rate 0.398402       0.358447 0.358447
  Balanced Accuracy 0.511725       0.500000 0.500000
----- Metrics for all Algorithms in Iteration 10 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
```

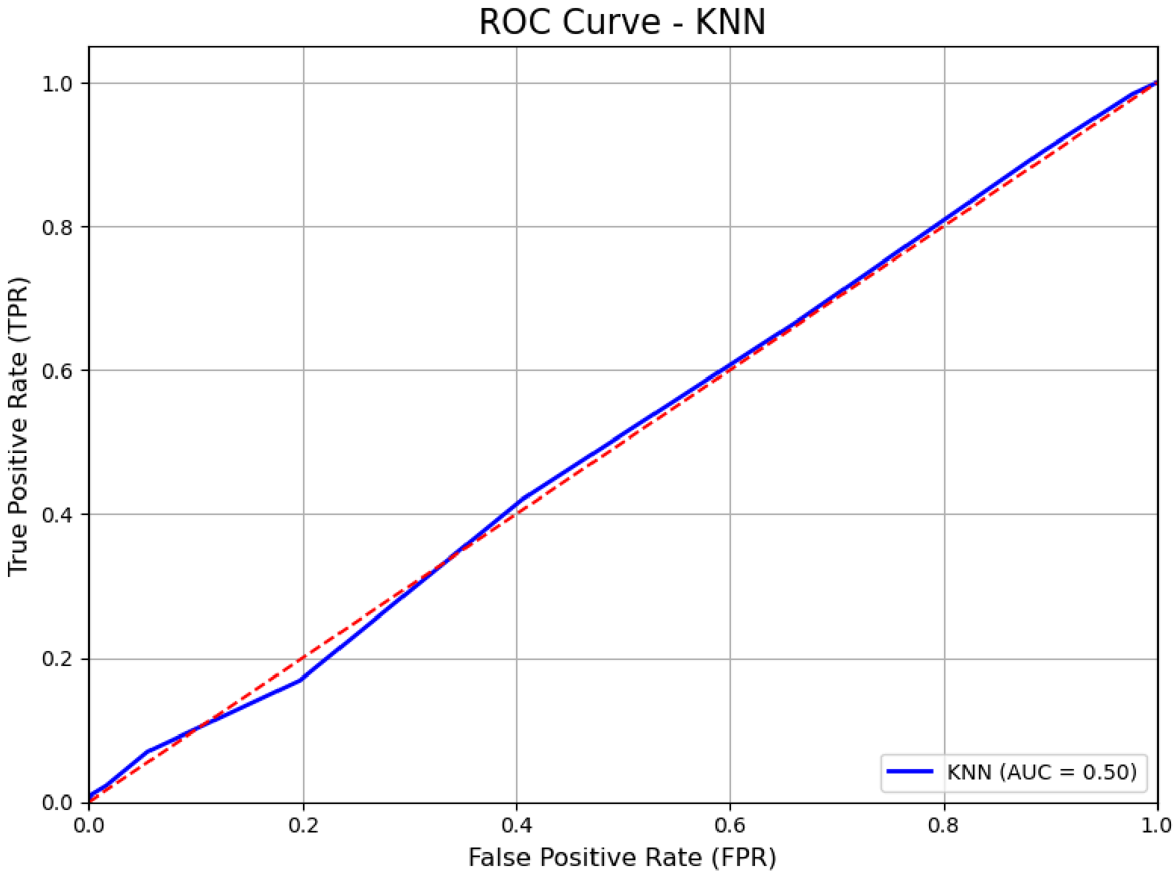
```
             Metric       KNN  Random Forest      LSTM
                FPR  0.197509       0.000000  0.001779
                FNR  0.831210       1.000000  1.000000
                TSS -0.028719       0.000000 -0.001779
                HSS -0.032103       0.000000 -0.002281
           Accuracy  0.575342       0.641553  0.640411
          Precision  0.323171       0.000000  0.000000
 Recall/Sensitivity  0.168790       0.000000  0.000000
        Specificity  0.802491       1.000000  0.998221
         F1 Measure  0.221757       0.000000  0.000000
         Error Rate  0.424658       0.358447  0.359589
  Balanced Accuracy  0.485640       0.500000  0.499110
```

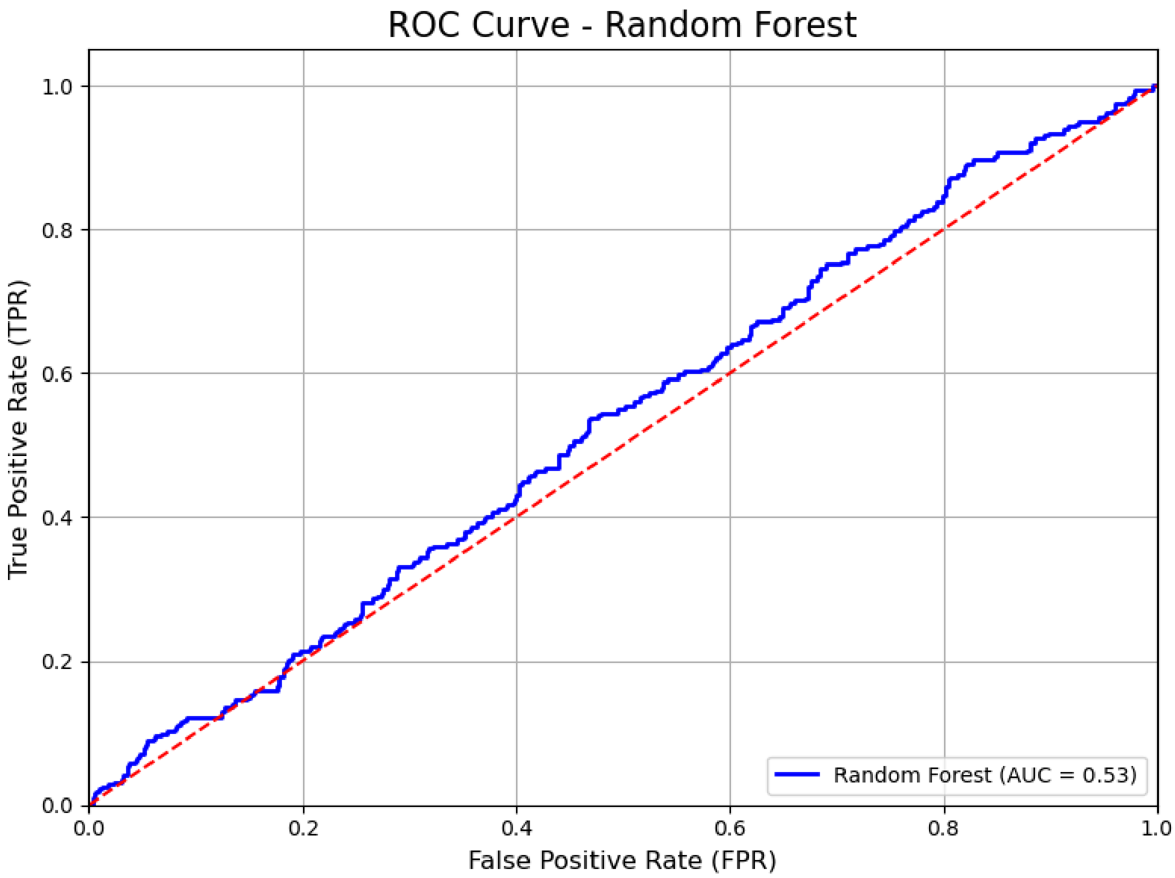
In [37]:

```
def plot_individual_roc_curve(y_test, y_pred_probs, model_name):
    """
    Plot the ROC curve for an individual model.

    Parameters:
        y_test (array-like): True labels for the test set.
        y_pred_probs (array-like): Predicted probabilities for the positive
        model_name (str): Name of the model.
    """
    # Calculate ROC curve
    fpr, tpr, _ = roc_curve(y_test, y_pred_probs)
    # Calculate AUC
    roc_auc = auc(fpr, tpr)

    # Create the plot
    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.2f})', color='
    plt.plot([0, 1], [0, 1], color='red', linestyle='--')  # Diagonal line
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate (FPR)', fontsize=12)
    plt.ylabel('True Positive Rate (TPR)', fontsize=12)
    plt.title(f'ROC Curve - {model_name}', fontsize=16)
    plt.legend(loc='lower right', fontsize=10)
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

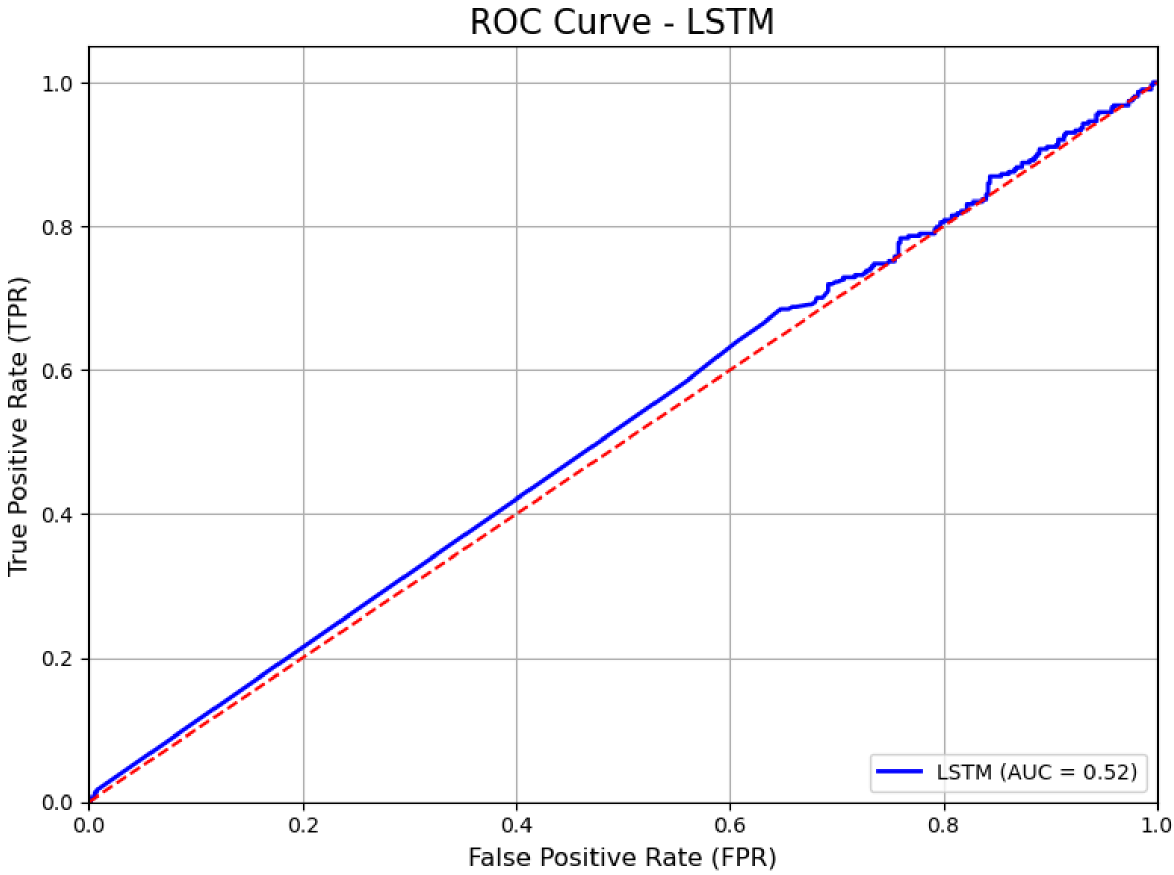
In [38]:

```
# ---- KNN ----
y_pred_KNN_probs = best_KNN.predict_proba(X_test)[:, 1]  # Probability for
plot_individual_roc_curve(y_test, y_pred_KNN_probs, 'KNN')

# ---- Random Forest ----
y_pred_randomforest_probs = best_randomforest.predict_proba(X_test)[:, 1]  #
plot_individual_roc_curve(y_test, y_pred_randomforest_probs, 'Random Forest

# ---- LSTM ----
y_pred_lstm_probs = lstm_model.predict(X_test_lstm)[:, 1]  # Probability fo
plot_individual_roc_curve(y_test, y_pred_lstm_probs, 'LSTM')
```






```
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

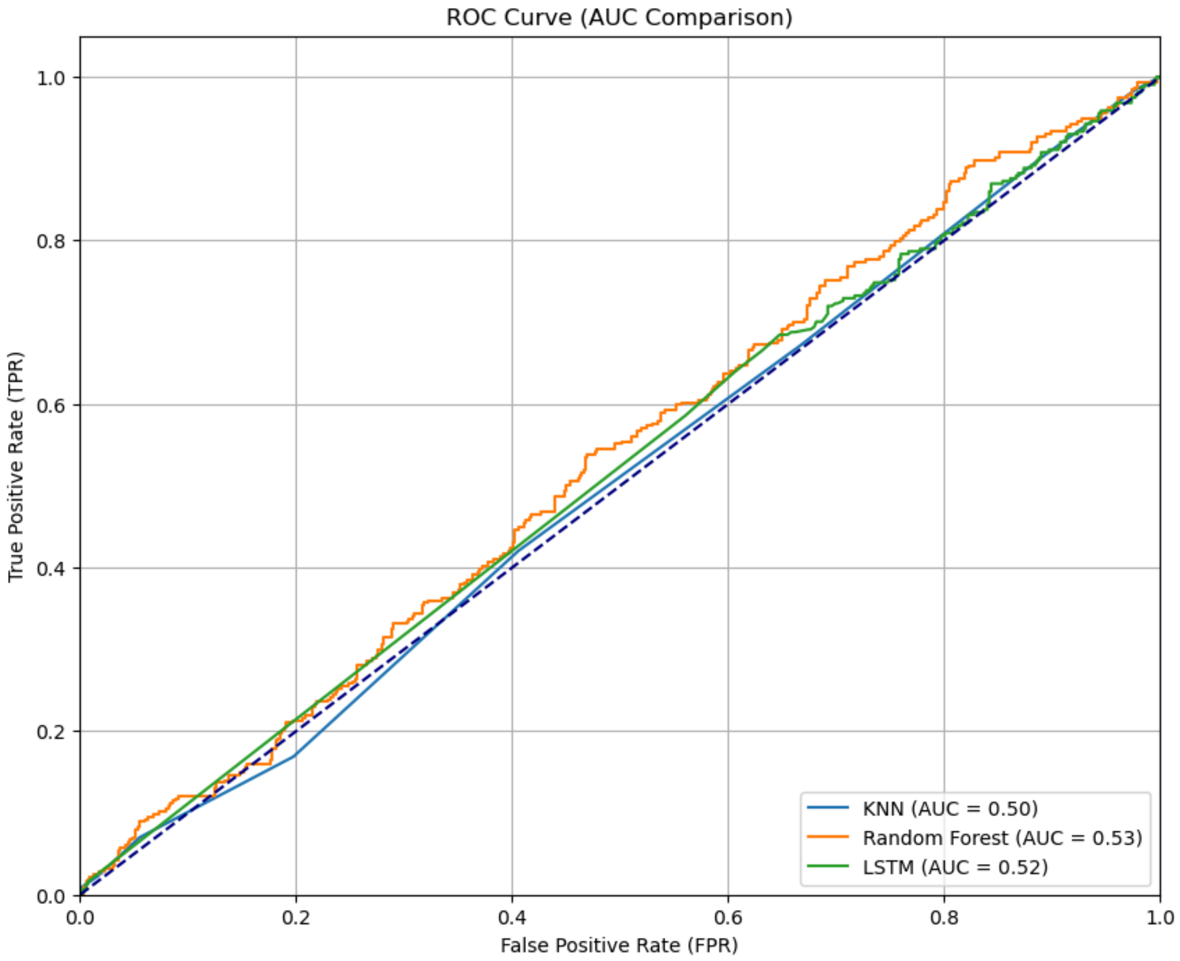


In [39]:

```
# Function to plot ROC curve and AUC
def plot_roc_curve(y_test, y_pred_probs, model_name):
    # Calculate ROC curve
    fpr, tpr, _ = roc_curve(y_test, y_pred_probs)
    # Calculate AUC
    roc_auc = auc(fpr, tpr)

    # Plot ROC curve
    plt.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.2f})')

# Initialize plot
plt.figure(figsize=(10, 8))

# ---- KNN ----
y_pred_knn_probs = best_KNN.predict_proba(X_test)[:, 1]  # Probability for c
plot_roc_curve(y_test, y_pred_knn_probs, 'KNN')

# ---- Random Forest ----
y_pred_rf_probs = best_randomforest.predict_proba(X_test)[:, 1]  # Probabili
plot_roc_curve(y_test, y_pred_rf_probs, 'Random Forest')

# ---- LSTM ----
y_pred_lstm_probs = lstm_model.predict(X_test_lstm)[:, 1]  # Probability fo
plot_roc_curve(y_test, y_pred_lstm_probs, 'LSTM')

# Customize plot
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')  # Diagonal line (ra
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve (AUC Comparison)')
plt.legend(loc='lower right')
plt.grid(True)
```

```
# Show the plot
plt.show()
```

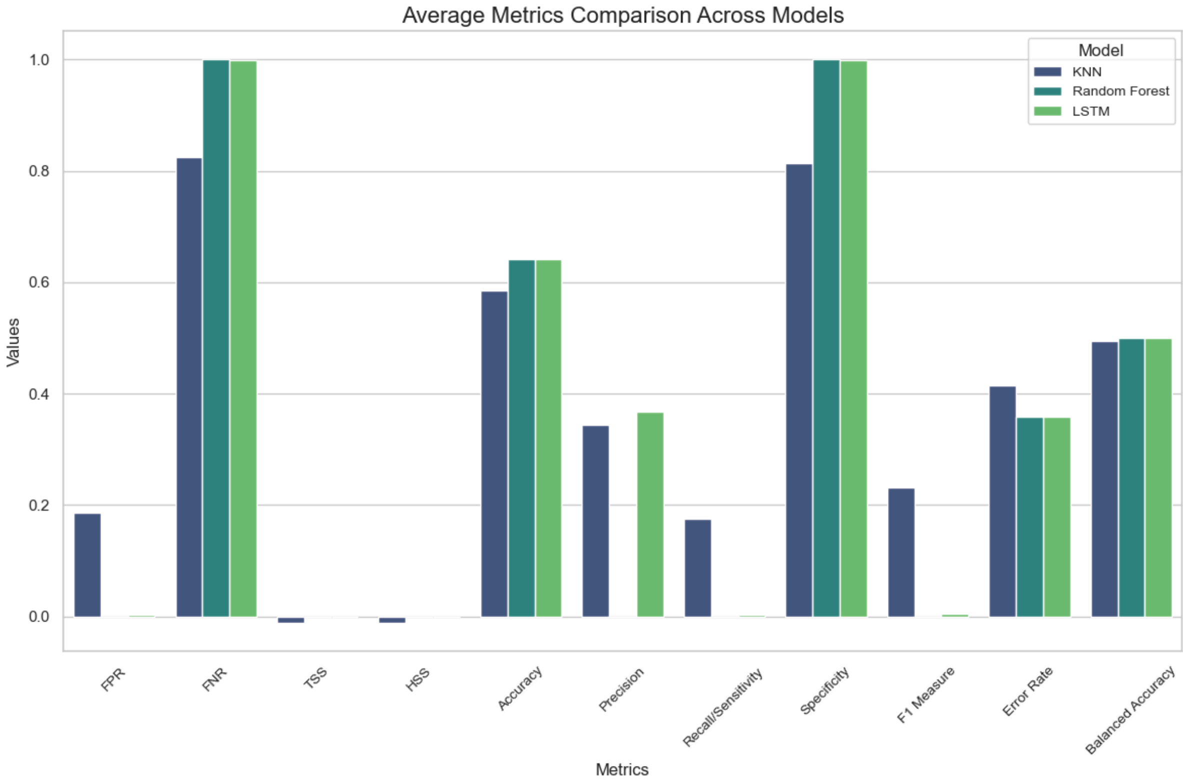
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step



In [40]:
```
# Initialize an empty dictionary to store average metrics for each model
average_metrics = {'Metric': []}

# Gather metrics for each model across folds
for model_name in ['KNN', 'Random Forest', 'LSTM']:
    # Extract all metric values for the current model from the folds
    metrics_by_model = {metric: [] for metric in all_metrics[0][model_name]}
    for fold_metrics in all_metrics:
        for metric, value in fold_metrics[model_name].items():
            metrics_by_model[metric].append(value)

    # Compute average metrics for the current model
    average_metrics['Metric'] = list(metrics_by_model.keys())
    average_metrics[model_name] = [sum(values) / len(values) for values in m

# Convert the average metrics dictionary to a DataFrame
avg_performance_df = pd.DataFrame(average_metrics)

# Print the DataFrame
print("Average Performance Across All Folds:")
print(avg_performance_df.round(decimals=2))

# Create a bar plot for the metrics comparison
plt.figure(figsize=(12, 8))
sbn.set_theme(style="whitegrid")

# Melt the DataFrame for plotting
melted_df = avg_performance_df.melt(id_vars='Metric', var_name='Model', valu
```

```
# Plot the data using seaborn
sbn.barplot(data=melted_df, x='Metric', y='Value', hue='Model', palette='vi

# Customize the plot
plt.title('Average Metrics Comparison Across Models', fontsize=16)
plt.xlabel('Metrics', fontsize=12)
plt.ylabel('Values', fontsize=12)
plt.xticks(rotation=45, fontsize=10)
plt.legend(title='Model', fontsize=10)
plt.tight_layout()

# Show the plot
plt.show()
```

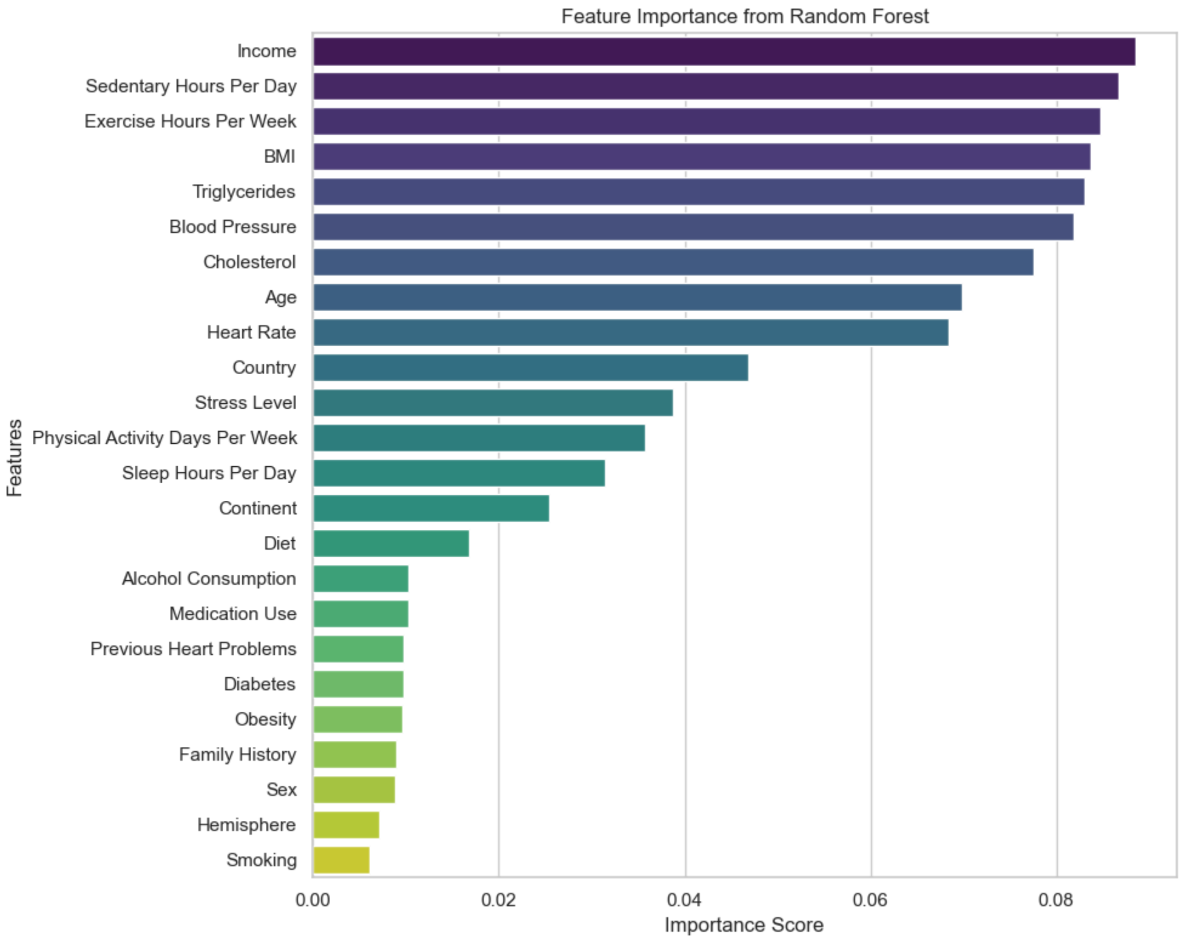
```
Average Performance Across All Folds:
                 Metric   KNN  Random Forest  LSTM
0                   FPR  0.19           0.00  0.00
1                   FNR  0.82           1.00  1.00
2                   TSS -0.01           0.00 -0.00
3                   HSS -0.01           0.00 -0.00
4              Accuracy  0.59           0.64  0.64
5             Precision  0.34           0.00  0.37
6    Recall/Sensitivity  0.18           0.00  0.00
7           Specificity  0.81           1.00  1.00
8            F1 Measure  0.23           0.00  0.00
9            Error Rate  0.41           0.36  0.36
10    Balanced Accuracy  0.49           0.50  0.50
```



In this experiment, we can observe that the models do not achieve a high accuracy. In general terms, the models predict pretty well the cases where there is no risk of a heart attack (the negative cases), but struggle to predict correctly the positive cases. This is pretty bad because a good model in the healthcare industry should predict and be focused when there is a risk of heart attack to try to prevent it.

In [41]:
```
# Random Forest Feature Selection
importances = best_randomforest.feature_importances_
feature_names = X.columns
```

```
# Create a DataFrame for better visualization
feature_importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Plot the feature importances
plt.figure(figsize=(10, 8))
sbn.barplot(x='Importance', y='Feature', data=feature_importance_df, palette
plt.title('Feature Importance from Random Forest')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.tight_layout()
plt.show()
```



We check for feature importance and we reduce the size of features in the training data, focusing on the most important ones.

In [42]:
```
# Select top 10 features
top_10_features = feature_importance_df.head(10)['Feature'].tolist()

# Update dataset with only the top 10 features
X_top_10 = X[top_10_features]

# Print selected features
print("Top 10 Features Selected:")
print(top_10_features)
```

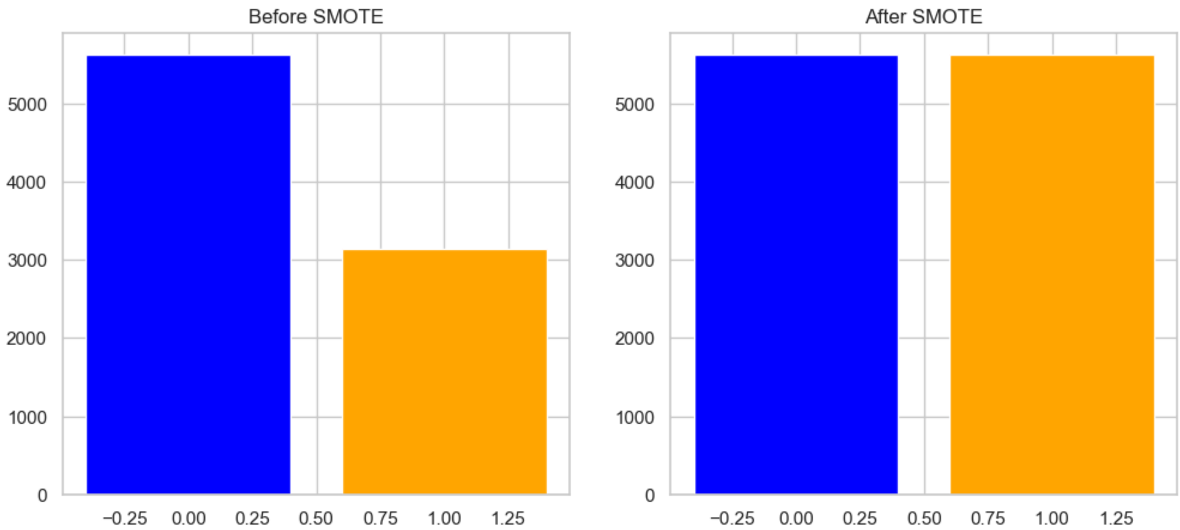
```
Top 10 Features Selected:
['Income', 'Sedentary Hours Per Day', 'Exercise Hours Per Week', 'BMI', 'Tr
iglycerides', 'Blood Pressure', 'Cholesterol', 'Age', 'Heart Rate', 'Countr
y']
```

As well we use SMOTE to balance the data since there is some class imbalance.

In [43]:
```
# Initialize SMOTE
smote = SMOTE(random_state=42)

# Apply SMOTE to the reduced dataset (X_top_10)
X_smote, y_smote = smote.fit_resample(X_top_10, y)

# Display the new class distribution
from collections import Counter
print(f"Class distribution after SMOTE: {Counter(y_smote)}")
```

```
Class distribution after SMOTE: Counter({0.0: 5624, 1.0: 5624})
```

In [44]:
```
# Plot class distribution before and after SMOTE
fig, ax = plt.subplots(1, 2, figsize=(12, 5))
ax[0].bar(Counter(y).keys(), Counter(y).values(), color=['blue', 'orange'])
ax[0].set_title('Before SMOTE')
ax[1].bar(Counter(y_smote).keys(), Counter(y_smote).values(), color=['blue',
ax[1].set_title('After SMOTE')
plt.show()
```



In [45]:
```
# Perform cross-validation with top 10 features
for fold, (train_index, test_index) in enumerate(kf.split(X_top_10, y)):
    print(f"----- Metrics for all Algorithms in Iteration {fold + 1} -----"

    # Split data
    X_train, X_test = X_top_10.iloc[train_index], X_top_10.iloc[test_index]
    y_train, y_test = y_smote.iloc[train_index], y_smote.iloc[test_index]

    # ---- Random Forest ----
    best_randomforest.fit(X_train, y_train)
    y_pred_randomforest = best_randomforest.predict(X_test)
    randomforest_metrics = calculate_metrics(y_test, y_pred_randomforest)

    # ---- KNN ----
    best_KNN.fit(X_train, y_train)
    y_pred_KNN = best_KNN.predict(X_test)
    KNN_metrics = calculate_metrics(y_test, y_pred_KNN)


    # ---- LSTM ----
    X_train_lstm = X_train.values.reshape(X_train.shape[0], 1, X_train.shape
    X_test_lstm = X_test.values.reshape(X_test.shape[0], 1, X_test.shape[1]
    y_train_lstm = to_categorical(y_train, num_classes=2)
    y_test_lstm = to_categorical(y_test, num_classes=2)
```

```
    lstm_model = Sequential([
        Input(shape=(1, X_train.shape[1])),
        LSTM(64, activation='relu'),
        Dropout(0.2),
        Dense(2, activation='softmax')
    ])
    lstm_model.compile(optimizer='adam', loss='categorical_crossentropy', me
    lstm_model.fit(X_train_lstm, y_train_lstm, epochs=10, batch_size=16, ve

    y_pred_lstm = np.argmax(lstm_model.predict(X_test_lstm), axis=1)
    lstm_metrics = calculate_metrics(y_test, y_pred_lstm)

    # ---- Combine Metrics ----
    all_metrics_fold = {
        'KNN': KNN_metrics,
        'Random Forest': randomforest_metrics,
        'LSTM': lstm_metrics
    }

    # Store metrics for the current fold
    all_metrics.append(all_metrics_fold)

    # ---- Print Metrics for All Algorithms in Current Iteration ----
    iteration_metrics = pd.DataFrame({
        'Metric': list(KNN_metrics.keys()),
        'KNN': list(KNN_metrics.values()),
        'Random Forest': list(randomforest_metrics.values()),
        'LSTM': list(lstm_metrics.values())
    })
    print(iteration_metrics.to_string(index=False))
```

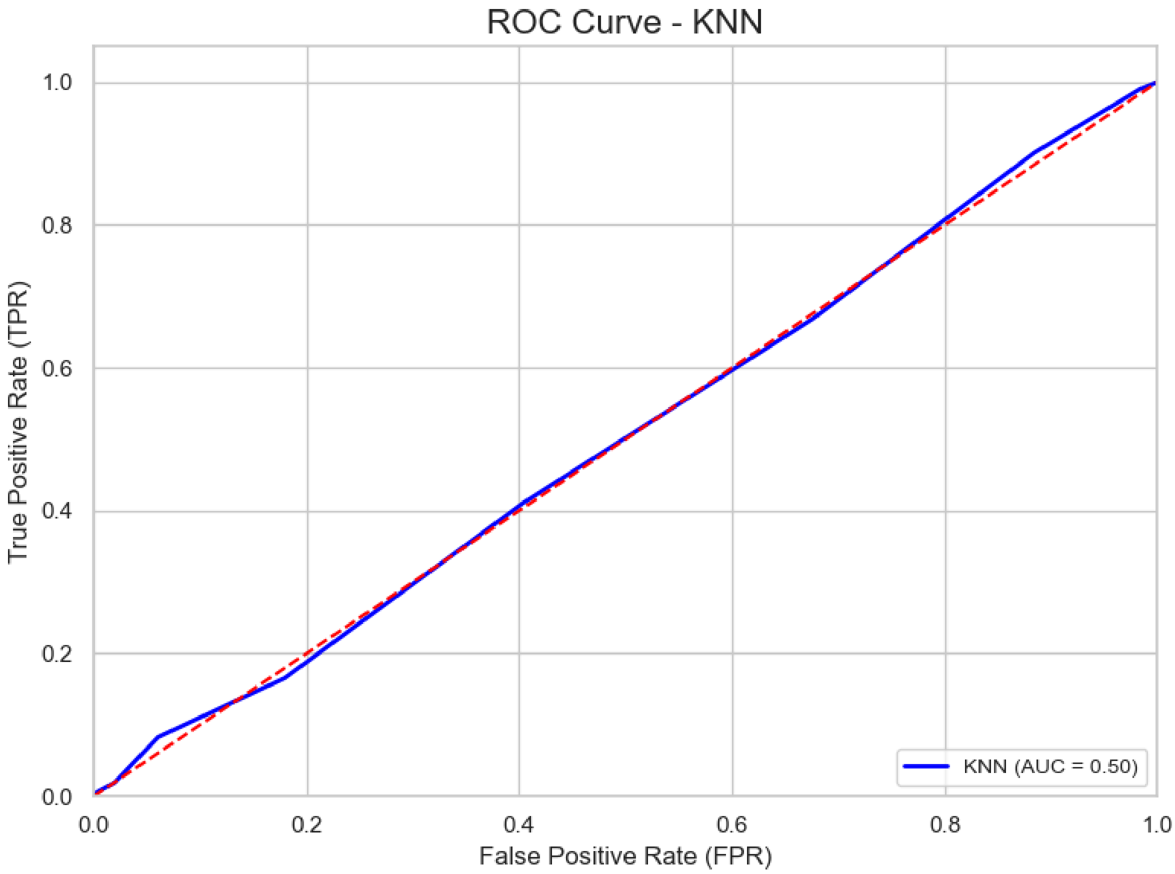
```
----- Metrics for all Algorithms in Iteration 1 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
            Metric       KNN  Random Forest      LSTM
               FPR  0.159858       0.001776  0.000000
               FNR  0.837580       1.000000  1.000000
               TSS  0.002562      -0.001776  0.000000
               HSS  0.002918      -0.002278  0.000000
          Accuracy  0.597491       0.640821  0.641961
         Precision  0.361702       0.000000  0.000000
Recall/Sensitivity  0.162420       0.000000  0.000000
       Specificity  0.840142       0.998224  1.000000
        F1 Measure  0.224176       0.000000  0.000000
        Error Rate  0.402509       0.359179  0.358039
 Balanced Accuracy  0.501281       0.499112  0.500000
----- Metrics for all Algorithms in Iteration 2 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step
            Metric        KNN  Random Forest      LSTM
               FPR   0.204263       0.000000  0.000000
               FNR   0.812102       1.000000  1.000000
               TSS  -0.016365       0.000000  0.000000
               HSS  -0.018155       0.000000  0.000000
          Accuracy   0.578107       0.641961  0.641961
         Precision   0.339080       0.000000  0.000000
Recall/Sensitivity   0.187898       0.000000  0.000000
       Specificity   0.795737       1.000000  1.000000
        F1 Measure   0.241803       0.000000  0.000000
        Error Rate   0.421893       0.358039  0.358039
 Balanced Accuracy   0.491818       0.500000  0.500000
----- Metrics for all Algorithms in Iteration 3 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step
            Metric        KNN  Random Forest      LSTM
               FPR   0.190053       0.000000  0.000000
               FNR   0.875796       1.000000  1.000000
               TSS  -0.065849       0.000000  0.000000
               HSS  -0.074686       0.000000  0.000000
          Accuracy   0.564424       0.641961  0.641961
         Precision   0.267123       0.000000  0.000000
Recall/Sensitivity   0.124204       0.000000  0.000000
       Specificity   0.809947       1.000000  1.000000
        F1 Measure   0.169565       0.000000  0.000000
        Error Rate   0.435576       0.358039  0.358039
 Balanced Accuracy   0.467075       0.500000  0.500000
----- Metrics for all Algorithms in Iteration 4 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
            Metric       KNN  Random Forest      LSTM
               FPR  0.170515       0.000000  0.001776
               FNR  0.805112       1.000000  0.996805
               TSS  0.024373       0.000000  0.001419
               HSS  0.027406       0.000000  0.001820
          Accuracy  0.602740       0.642694  0.642694
         Precision  0.388535       0.000000  0.500000
Recall/Sensitivity  0.194888       0.000000  0.003195
       Specificity  0.829485       1.000000  0.998224
        F1 Measure  0.259574       0.000000  0.006349
        Error Rate  0.397260       0.357306  0.357306
 Balanced Accuracy  0.512187       0.500000  0.500709
----- Metrics for all Algorithms in Iteration 5 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step
            Metric       KNN  Random Forest      LSTM
               FPR  0.185053       0.000000  0.000000
               FNR  0.786624       1.000000  1.000000
               TSS  0.028322       0.000000  0.000000
               HSS  0.031486       0.000000  0.000000
          Accuracy  0.599315       0.641553  0.641553
```

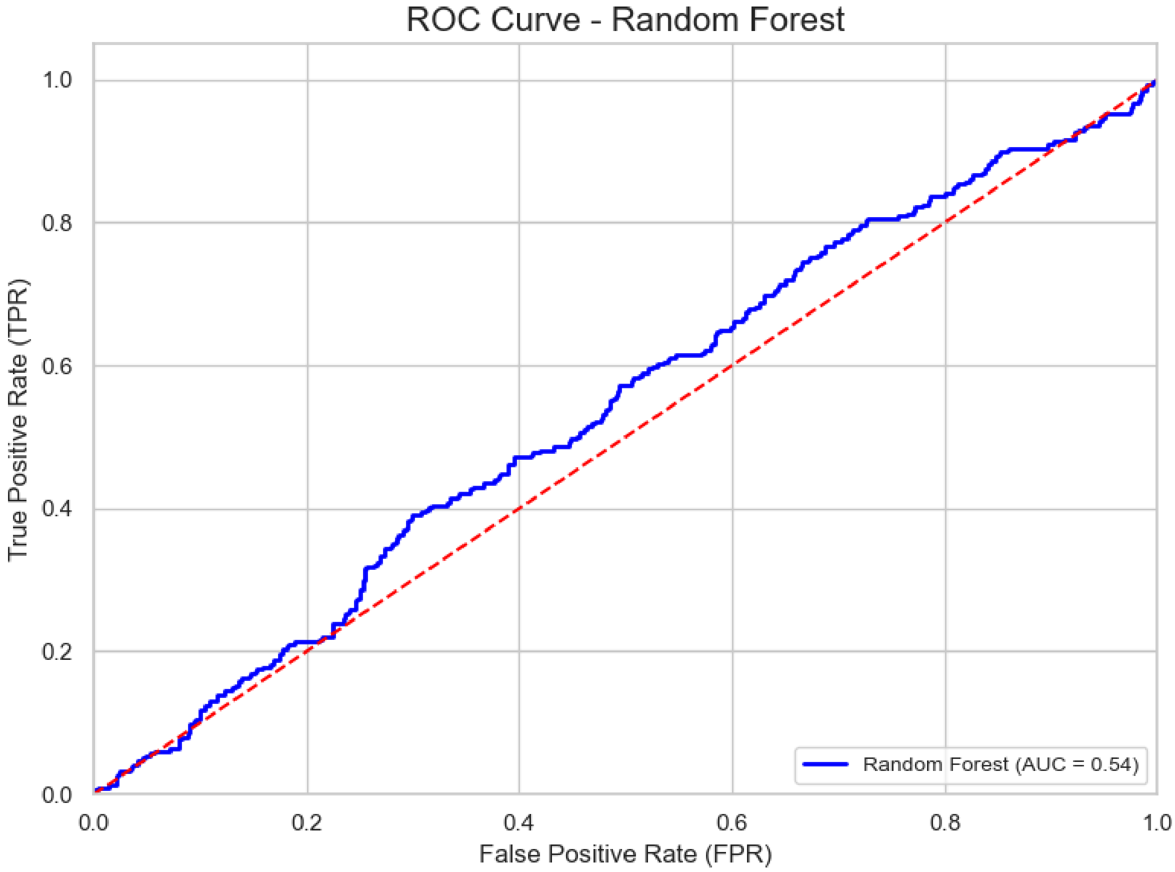
```
         Precision 0.391813       0.000000 0.000000
Recall/Sensitivity 0.213376       0.000000 0.000000
       Specificity 0.814947       1.000000 1.000000
        F1 Measure 0.276289       0.000000 0.000000
        Error Rate 0.400685       0.358447 0.358447
 Balanced Accuracy 0.514161       0.500000 0.500000
----- Metrics for all Algorithms in Iteration 6 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 32ms/step
            Metric       KNN  Random Forest     LSTM
               FPR  0.201068       0.001779 0.000000
               FNR  0.828025       1.000000 1.000000
               TSS -0.029093      -0.001779 0.000000
               HSS -0.032444      -0.002281 0.000000
          Accuracy  0.574201       0.640411 0.641553
         Precision  0.323353       0.000000 0.000000
Recall/Sensitivity  0.171975       0.000000 0.000000
       Specificity  0.798932       0.998221 1.000000
        F1 Measure  0.224532       0.000000 0.000000
        Error Rate  0.425799       0.359589 0.358447
 Balanced Accuracy  0.485453       0.499110 0.500000
----- Metrics for all Algorithms in Iteration 7 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step
            Metric       KNN  Random Forest     LSTM
               FPR  0.197509       0.003559 0.000000
               FNR  0.815287       1.000000 1.000000
               TSS -0.012796      -0.003559 0.000000
               HSS -0.014247      -0.004558 0.000000
          Accuracy  0.581050       0.639269 0.641553
         Precision  0.343195       0.000000 0.000000
Recall/Sensitivity  0.184713       0.000000 0.000000
       Specificity  0.802491       0.996441 1.000000
        F1 Measure  0.240166       0.000000 0.000000
        Error Rate  0.418950       0.360731 0.358447
 Balanced Accuracy  0.493602       0.498221 0.500000
----- Metrics for all Algorithms in Iteration 8 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step
            Metric       KNN  Random Forest      LSTM
               FPR  0.160142       0.001779  0.003559
               FNR  0.847134       1.000000  0.996815
               TSS -0.007276      -0.001779 -0.000374
               HSS -0.008303      -0.002281 -0.000479
          Accuracy  0.593607       0.640411  0.640411
         Precision  0.347826       0.000000  0.333333
Recall/Sensitivity  0.152866       0.000000  0.003185
       Specificity  0.839858       0.998221  0.996441
        F1 Measure  0.212389       0.000000  0.006309
        Error Rate  0.406393       0.359589  0.359589
 Balanced Accuracy  0.496362       0.499110  0.499813
----- Metrics for all Algorithms in Iteration 9 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step
            Metric      KNN  Random Forest     LSTM
               FPR 0.160142       0.000000 0.000000
               FNR 0.805732       1.000000 1.000000
               TSS 0.034125       0.000000 0.000000
               HSS 0.038539       0.000000 0.000000
          Accuracy 0.608447       0.641553 0.641553
         Precision 0.403974       0.000000 0.000000
Recall/Sensitivity 0.194268       0.000000 0.000000
       Specificity 0.839858       1.000000 1.000000
        F1 Measure 0.262366       0.000000 0.000000
        Error Rate 0.391553       0.358447 0.358447
 Balanced Accuracy 0.517063       0.500000 0.500000
----- Metrics for all Algorithms in Iteration 10 -----
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step
```

```
              Metric       KNN  Random Forest      LSTM
                 FPR  0.179715       0.000000  0.000000
                 FNR  0.834395       0.996815  1.000000
                 TSS -0.014110       0.003185  0.000000
                 HSS -0.015910       0.004083  0.000000
            Accuracy  0.585616       0.642694  0.641553
           Precision  0.339869       1.000000  0.000000
  Recall/Sensitivity  0.165605       0.003185  0.000000
         Specificity  0.820285       1.000000  1.000000
          F1 Measure  0.222698       0.006349  0.000000
          Error Rate  0.414384       0.357306  0.358447
   Balanced Accuracy  0.492945       0.501592  0.500000
```

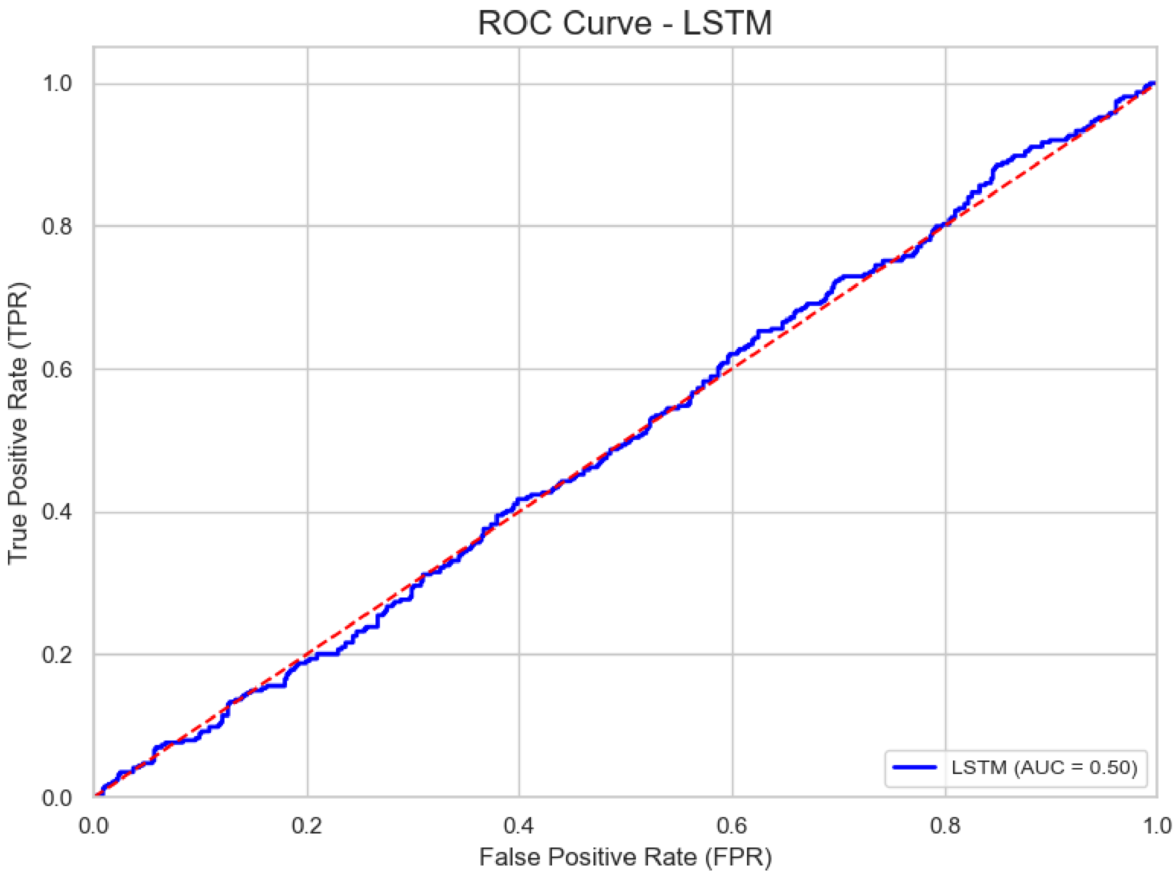
In [48]:

```
# ---- KNN ----
y_pred_KNN_probs = best_KNN.predict_proba(X_test)[:, 1]  # Probability for c
plot_individual_roc_curve(y_test, y_pred_KNN_probs, 'KNN')

# ---- Random Forest ----
y_pred_randomforest_probs = best_randomforest.predict_proba(X_test)[:, 1]  #
plot_individual_roc_curve(y_test, y_pred_randomforest_probs, 'Random Forest

# ---- LSTM ----
y_pred_lstm_probs = lstm_model.predict(X_test_lstm)[:, 1]  # Probability for
plot_individual_roc_curve(y_test, y_pred_lstm_probs, 'LSTM')
```






```
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

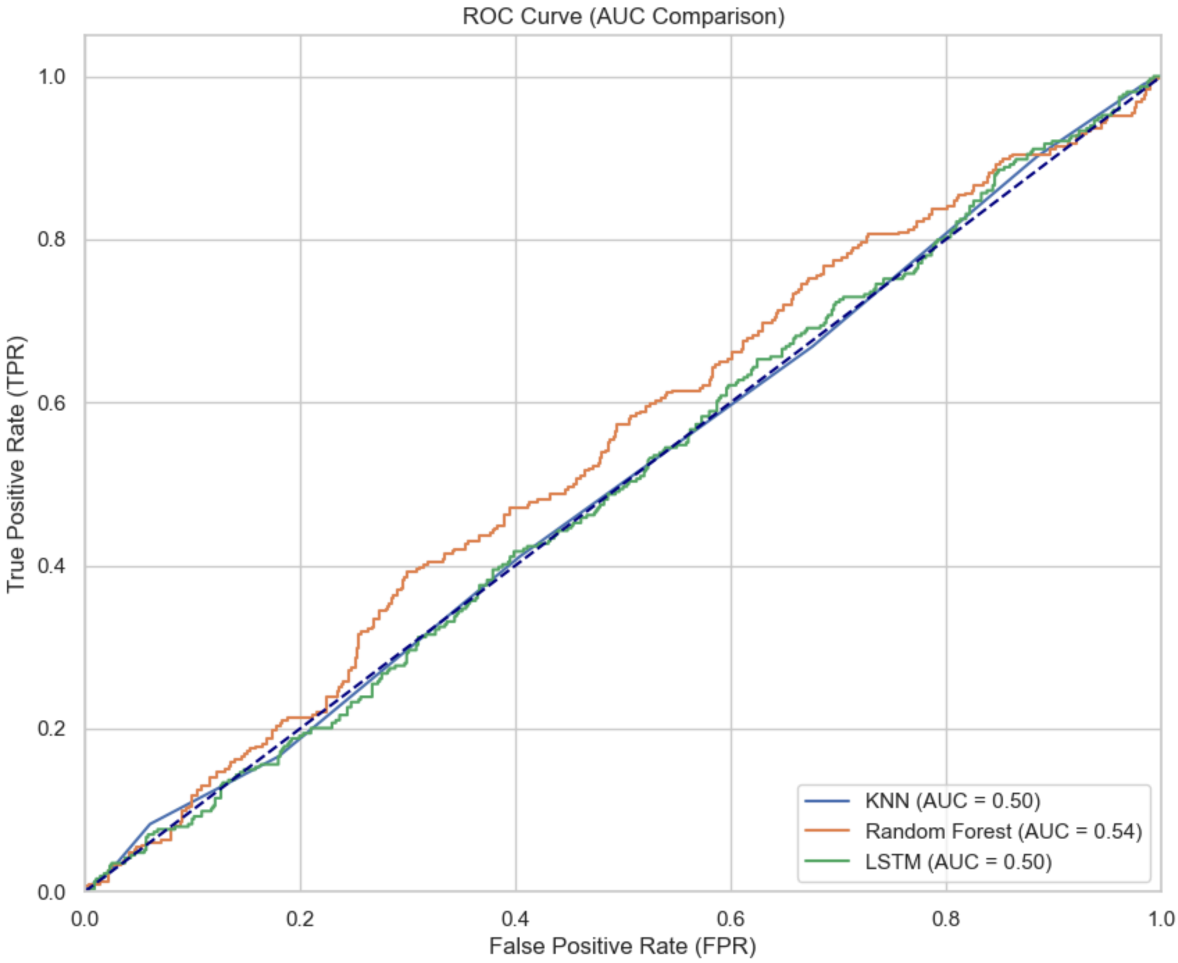


In [47]:

```python
# Initialize plot
plt.figure(figsize=(10, 8))

# ---- KNN ----
y_pred_knn_probs = best_KNN.predict_proba(X_test)[:, 1]  # Probability for c
plot_roc_curve(y_test, y_pred_knn_probs, 'KNN')
```

```
# ---- Random Forest ----
y_pred_rf_probs = best_randomforest.predict_proba(X_test)[:, 1]  # Probabili
plot_roc_curve(y_test, y_pred_rf_probs, 'Random Forest')

# ---- LSTM ----
y_pred_lstm_probs = lstm_model.predict(X_test_lstm)[:, 1]  # Probability fo
plot_roc_curve(y_test, y_pred_lstm_probs, 'LSTM')

# Customize plot
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')  # Diagonal line (ra
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve (AUC Comparison)')
plt.legend(loc='lower right')
plt.grid(True)

# Show the plot
plt.show()
```

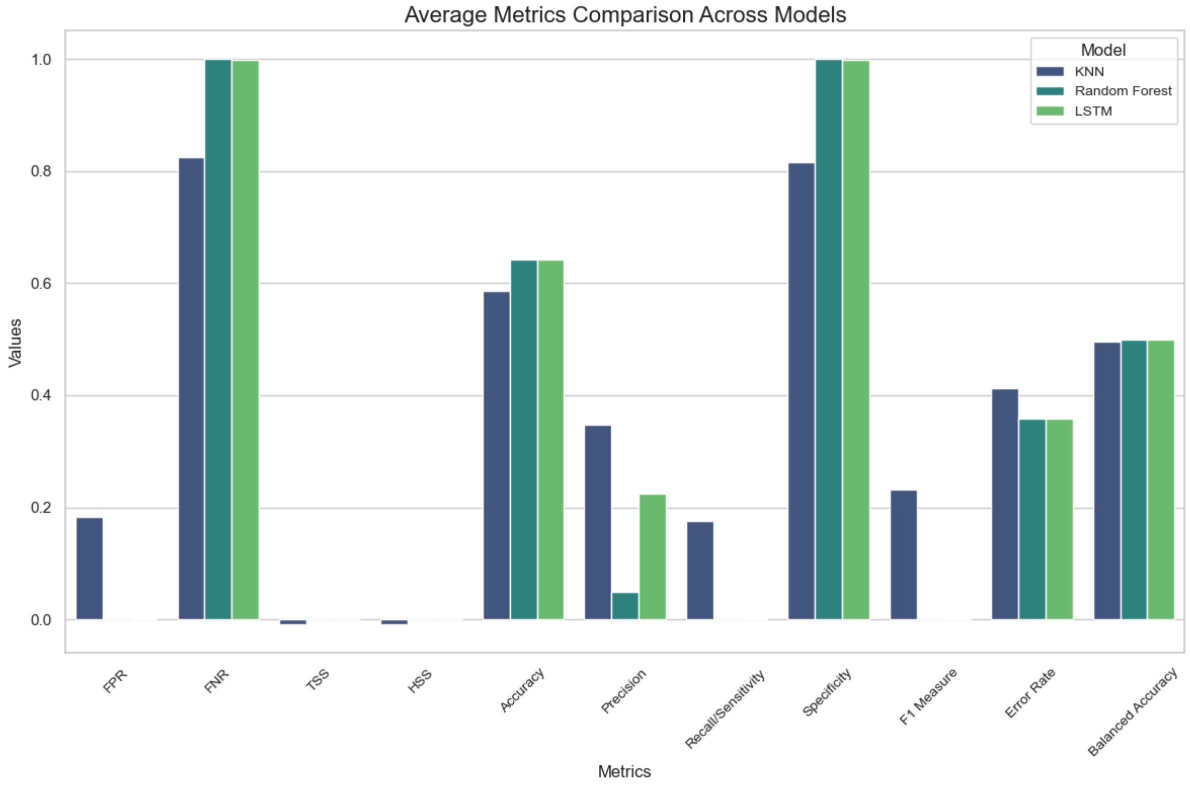
```
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```



In [46]:
```
# Initialize an empty dictionary to store average metrics for each model
average_metrics = {'Metric': []}

# Gather metrics for each model across folds
for model_name in ['KNN', 'Random Forest', 'LSTM']:
    # Extract all metric values for the current model from the folds
    metrics_by_model = {metric: [] for metric in all_metrics[0][model_name]
    for fold_metrics in all_metrics:
        for metric, value in fold_metrics[model_name].items():
            metrics_by_model[metric].append(value)
```

```python
    # Compute average metrics for the current model
    average_metrics['Metric'] = list(metrics_by_model.keys())
    average_metrics[model_name] = [sum(values) / len(values) for values in m

# Convert the average metrics dictionary to a DataFrame
avg_performance_df = pd.DataFrame(average_metrics)

# Print the DataFrame
print("Average Performance Across All Folds:")
print(avg_performance_df.round(decimals=2))

# Create a bar plot for the metrics comparison
plt.figure(figsize=(12, 8))
sbn.set_theme(style="whitegrid")

# Melt the DataFrame for plotting
melted_df = avg_performance_df.melt(id_vars='Metric', var_name='Model', valu

# Plot the data using seaborn
sbn.barplot(data=melted_df, x='Metric', y='Value', hue='Model', palette='vi

# Customize the plot
plt.title('Average Metrics Comparison Across Models', fontsize=16)
plt.xlabel('Metrics', fontsize=12)
plt.ylabel('Values', fontsize=12)
plt.xticks(rotation=45, fontsize=10)
plt.legend(title='Model', fontsize=10)
plt.tight_layout()

# Show the plot
plt.show()
```

```
Average Performance Across All Folds:
                 Metric   KNN  Random Forest  LSTM
0                   FPR  0.18           0.00  0.00
1                   FNR  0.82           1.00  1.00
2                   TSS -0.01          -0.00  0.00
3                   HSS -0.01          -0.00  0.00
4              Accuracy  0.59           0.64  0.64
5             Precision  0.35           0.05  0.22
6    Recall/Sensitivity  0.18           0.00  0.00
7           Specificity  0.82           1.00  1.00
8            F1 Measure  0.23           0.00  0.00
9            Error Rate  0.41           0.36  0.36
10    Balanced Accuracy  0.50           0.50  0.50
```



In conclusion, even though we balanced the data and reduce the dimension of the training features, the models did not improve their performance. Therefore, for the future we probably should work a bit more on the data preprocessing to see if we can get more relevant data.

Side Notes

To be able to run this code make sure to have installed the following libraries: pandas, numpy, seaborn, matplotlib, sklearn, tensorflow and imbalanced-learn.
To install the libraries write the following in the terminal: pip install (name of the library).
Another way to install the libraries is to write in a cell of the jupiter notebook: !pip install (name of the library).
Moreover, it is important to have the required csv file, jupiter notebook, and python file in the same folder in order to run the codes.

https://github.com/psalarc/salar_pablo_finaltermproj