Save, Load and K-Fold Validation

RandomForest= 80.26% accuracy(kfold)

SVC=52.63% accuracy(kfold), 88.61% accuracy(pipeline)

DecisionTreeClassifier=61.84% accuracy (kfold)

```
import librosa
import soundfile
import os, glob, pickle
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
```

```
#load data set
arr=np.load('/content/drive/MyDrive/#/test.npy')
arr1=np.load('/content/drive/MyDrive/#/test1.npy')
```

```
#conversion to dataframe
DF = pd.DataFrame(arr)
DF1=pd.DataFrame(arr1)
```

```
#conversion to csv
DF.to_csv("data1.csv")
DF1.to_csv("data2.csv")
```

```
d=pd.read_csv('/content/data1.csv')
d1=pd.read_csv('/content/data2.csv')
```

```
d=d.values
d1=d1.values
```

```
X=d[:,0:40]
Y=d1[:,1]
```

```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
scores = cross_val_score(rf, X, Y, cv=10, scoring='accuracy')
print(scores.mean())
```

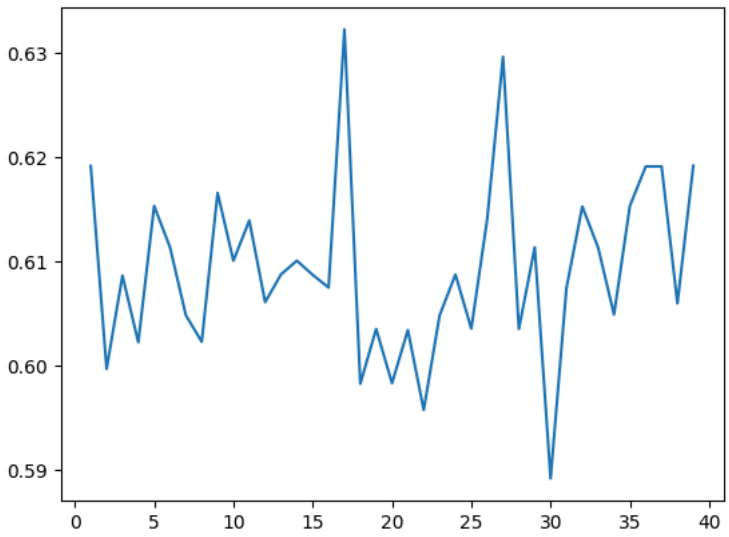
```
0.5968045112781956
```

```
rf_range = list(range(1, 40))
rf_scores = []
for k in rf_range:
    rf = RandomForestClassifier()
    scores = cross_val_score(rf, X, Y, cv=10, scoring='accuracy')
    rf_scores.append(scores.mean())
print(rf_scores)
```

```
[0.619172932330827, 0.5996582365003418, 0.6086295283663705, 0.6022214627477787, 0.6153280929596721, 0.6113465481886534, 0.6048
```

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(rf_range, rf_scores)
```

```
[<matplotlib.lines.Line2D at 0x7fcbc5b32440>]
```




```
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
clf = make_pipeline(StandardScaler(), SVC(gamma='auto'))
a=clf.fit(X, Y)


p=a.predict(X)


accuracy=accuracy_score(y_true=Y, y_pred=p)


print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 88.61%
```

```
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
cp=Pipeline([('scalar3',StandardScaler()),
('pca3',PCA(n_components=2)),
('rf_classifier',RandomForestClassifier(random_state=0))])
a=cp.fit(X, Y)


p=a.predict(X)


accuracy=accuracy_score(y_true=Y, y_pred=p)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 100.00%
```

```
from sklearn.tree import DecisionTreeClassifier
pipeline_dt=Pipeline([('scalar2',StandardScaler()),
('pca2',PCA(n_components=4)),
('dt_classifier',DecisionTreeClassifier(random_state=37))])


a=cp.fit(X, Y)


p=a.predict(X)


accuracy=accuracy_score(y_true=Y, y_pred=p)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 100.00%
```

```
X=d[:,0:40]
Y=d1[:,1]


Y
```

```
array(['calm', 'calm', 'calm', 'happy', 'calm', 'calm', 'calm', 'calm',
       'happy', 'happy', 'happy', 'happy', 'happy', 'happy', 'happy',
       'disgust', 'disgust', 'fearful', 'fearful', 'disgust', 'disgust',
       'disgust', 'fearful', 'fearful', 'fearful', 'disgust', 'fearful',
       'fearful', 'fearful', 'disgust', 'disgust', 'calm', 'calm', 'calm',
       'calm', 'calm', 'calm', 'calm', 'happy', 'happy', 'happy',
       'fearful', 'fearful', 'happy', 'fearful', 'happy', 'happy',
       'happy', 'calm', 'happy', 'fearful', 'fearful', 'fearful',
       'disgust', 'fearful', 'fearful', 'disgust', 'disgust', 'disgust',
       'disgust', 'disgust', 'disgust', 'disgust', 'calm', 'happy',
       'happy', 'happy', 'calm', 'calm', 'happy', 'calm', 'happy',
       'happy', 'calm', 'happy', 'calm', 'calm', 'calm', 'happy',
       'fearful', 'fearful', 'fearful', 'fearful', 'fearful', 'disgust',
       'disgust', 'disgust', 'fearful', 'disgust', 'disgust', 'disgust',
       'disgust', 'disgust', 'fearful', 'fearful', 'happy', 'calm',
       'happy', 'calm', 'calm', 'happy', 'happy', 'happy', 'happy',
       'calm', 'happy', 'happy', 'calm', 'calm', 'calm', 'calm',
       'disgust', 'fearful', 'disgust', 'fearful', 'disgust', 'disgust',
       'fearful', 'fearful', 'disgust', 'fearful', 'disgust', 'disgust',
       'fearful', 'fearful', 'fearful', 'disgust', 'happy', 'happy',
       'happy', 'calm', 'calm', 'happy', 'happy', 'calm', 'happy', 'calm',
       'happy', 'calm', 'calm', 'calm', 'happy', 'disgust', 'fearful',
       'disgust', 'fearful', 'disgust', 'disgust', 'disgust', 'fearful',
       'fearful', 'disgust', 'disgust', 'disgust', 'fearful', 'fearful',
       'fearful', 'fearful', 'calm', 'calm', 'calm', 'happy', 'happy',
       'calm', 'calm', 'calm', 'calm', 'happy', 'calm', 'happy', 'happy',
       'fearful', 'disgust', 'fearful', 'happy', 'disgust', 'disgust',
       'fearful', 'fearful', 'disgust', 'fearful', 'disgust', 'fearful',
       'happy', 'fearful', 'disgust', 'fearful', 'happy', 'disgust',
       'disgust', 'calm', 'calm', 'calm', 'calm', 'calm', 'disgust',
       'disgust', 'calm', 'happy', 'calm', 'fearful', 'happy', 'disgust',
       'fearful', 'disgust', 'fearful', 'happy', 'happy', 'fearful',
       'happy', 'fearful', 'fearful', 'disgust', 'disgust', 'disgust',
       'fearful', 'happy', 'calm', 'disgust', 'happy', 'fearful', 'happy',
       'calm', 'disgust', 'fearful', 'fearful', 'happy', 'fearful',
       'calm', 'happy', 'calm', 'calm', 'disgust', 'fearful', 'calm',
       'happy', 'happy', 'happy', 'calm', 'calm', 'fearful', 'happy',
       'fearful', 'disgust', 'disgust', 'fearful', 'happy', 'happy',
       'calm', 'fearful', 'disgust', 'disgust', 'disgust', 'disgust',
       'calm', 'calm', 'fearful', 'fearful', 'fearful', 'happy',
       'fearful', 'fearful', 'happy', 'calm', 'calm', 'calm', 'fearful',
       'calm', 'happy', 'calm', 'happy', 'happy', 'happy', 'happy',
       'calm', 'fearful', 'happy', 'disgust', 'disgust', 'disgust',
       'disgust', 'disgust', 'fearful', 'disgust', 'disgust', 'disgust',
       'calm', 'happy', 'calm', 'happy', 'calm', 'happy', 'calm', 'calm',
       'calm', 'calm', 'calm', 'happy', 'happy', 'happy', 'happy',
       'happy', 'disgust', 'fearful', 'fearful', 'disgust', 'disgust',
       'disgust', 'fearful', 'disgust', 'fearful', 'disgust', 'disgust',
       'fearful', 'disgust', 'fearful', 'fearful', 'fearful', 'calm',
       'calm', 'calm', 'calm', 'calm', 'fearful', 'fearful', 'disgust',
       'happy', 'disgust', 'fearful', 'fearful', 'fearful', 'happy',
       'calm', 'happy', 'happy', 'fearful', 'disgust', 'happy', 'happy',
       'calm', 'disgust', 'happy', 'calm', 'happy', 'disgust', 'fearful',
       'disgust', 'fearful', 'disgust', 'disgust', 'happy', 'disgust',
       'disgust', 'disgust', 'fearful', 'fearful', 'happy', 'fearful',
       'happy', 'disgust', 'calm', 'calm', 'happy', 'calm', 'calm',
       'calm', 'disgust', 'calm', 'happy', 'disgust', 'fearful',
       'fearful', 'calm', 'disgust', 'happy', 'fearful', 'happy',
```

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=5, shuffle=True, random_state=37)
```

```
print(kf)
```

```
KFold(n_splits=5, random_state=37, shuffle=True)
```

```
dg=KFold(n_splits=5, random_state=None, shuffle=False)
```

```
for i, (train_index, test_index) in enumerate(kf.split(X)):
    print(f"Fold {i}:")
    print(f"  Train: index={train_index}")
    print(f"  Test:  index={test_index}")
```

```
 297 302 307 311 313 314 319 326 331 332 340 343 347 348 350 351 352 359
 360 371 374 376 377 378 380 384 390 396 402 405 416 418 425 428 429 453
 455 457 458 461 463 464 474 479 487 490 499 506 515 517 520 524 529 532
 535 547 553 555 557 573 584 595 604 609 610 614 620 627 629 630 639 640
 642 650 654 655 657 674 679 682 683 690 691 693 705 710 712 715 719 725
 728 729 732 737 739 741 743 746 759]
Fold 4:
  Train: index=[  0   1   2   3   4   5   6   8  10  11  12  14  16  17  18  20  21  23
  24  25  26  27  28  30  31  32  34  36  37  38  39  40  42  43  44  47
  48  49  50  52  53  54  55  56  59  60  61  63  65  66  67  68  70  71
  72  74  75  76  77  78  80  81  82  83  84  86  88  89  90  91  92  93
  94  95  96  97  98  99 100 101 102 103 104 105 107 108 109 110 111 112
 113 114 117 118 120 121 122 123 124 125 126 127 129 130 131 132 133 134
 135 136 138 139 140 141 142 143 144 145 146 147 148 150 151 152 153 154
 155 156 157 158 160 161 162 163 164 165 166 168 169 171 172 173 174 175
 176 177 178 179 180 181 182 183 185 186 187 188 190 191 193 194 195 196
 197 198 200 201 203 204 205 206 207 208 209 210 211 212 213 214 215 216
 217 218 219 221 222 223 224 226 227 229 230 231 232 233 235 236 237 238
 239 240 241 242 243 245 247 248 249 252 253 255 256 257 258 259 260 261
 262 263 264 265 266 267 268 269 270 271 272 274 275 276 277 278 279 280
 282 283 284 285 286 288 289 290 291 293 294 296 297 298 301 302 303 304
 307 308 310 311 312 313 314 315 318 319 320 322 323 324 325 326 327 328
 329 330 331 332 334 335 337 338 339 340 341 342 343 346 347 348 349 350
 351 352 353 354 357 358 359 360 362 364 366 368 369 371 372 373 374 376
 377 378 379 380 381 382 383 384 386 388 389 390 392 393 395 396 397 400
 401 402 403 405 406 407 408 409 410 411 416 417 418 420 421 422 424 425
 428 429 430 431 432 433 436 438 440 441 442 443 445 446 447 448 449 450
 453 455 457 458 460 461 462 463 464 465 466 467 468 469 470 471 472 473
 474 475 476 479 480 481 483 484 485 486 487 489 490 491 492 493 494 495
 496 498 499 500 501 502 503 506 507 509 510 513 514 515 516 517 518 520
 521 522 524 525 526 527 528 529 530 531 532 533 534 535 537 538 539 541
 542 543 544 545 547 548 549 550 551 552 553 555 556 557 558 559 560 563
 564 567 569 572 573 574 576 577 578 579 580 581 582 583 584 585 586 588
 590 591 592 593 594 595 597 600 601 604 605 606 607 608 609 610 612 613
 614 615 616 617 618 620 621 622 623 624 625 626 627 628 629 630 632 633
 635 636 637 639 640 642 643 644 645 646 647 648 649 650 651 653 654 655
 656 657 658 659 660 662 663 665 666 667 669 672 673 674 676 677 678 679
 680 681 682 683 684 685 686 687 689 690 691 692 693 694 695 697 698 699
 700 701 702 703 704 705 706 707 708 710 712 713 714 715 718 719 720 721
 722 723 724 725 726 728 729 731 732 733 734 735 736 737 738 739 740 741
 742 743 745 746 747 748 750 751 752 753 754 755 756 758 759 760 761 763]
  Test:  index=[  7   9  13  15  19  22  29  33  35  41  45  46  51  57  58  62  64  69
  73  79  85  87 106 115 116 119 128 137 149 159 167 170 184 189 192 199
 202 220 225 228 234 244 246 250 251 254 273 281 287 292 295 299 300 305
 306 309 316 317 321 333 336 344 345 355 356 361 363 365 367 370 375 385
 387 391 394 398 399 404 412 413 414 415 419 423 426 427 434 435 437 439
 444 451 452 454 456 459 477 478 482 488 497 504 505 508 511 512 519 523
 536 540 546 554 561 562 565 566 568 570 571 575 587 589 596 598 599 602
 603 611 619 631 634 638 641 652 661 664 668 670 671 675 688 696 709 711
 716 717 727 730 744 749 757 762]
```

```
#kFold VAlidation
for train_index, test_index in kf.split(X, Y):
    print(f'TRAIN: {train_index}, TEST: {test_index}')
```

```
TRAIN: [  1   3   6   7   8   9  10  12  13  15  16  17  19  20  22  23  24  25
  26  27  29  31  33  35  36  38  39  40  41  42  43  44  45  46  47  50
  51  52  53  54  55  57  58  60  61  62  64  66  67  68  69  70  71  72
  73  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91
  92  95  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114
 115 116 117 118 119 120 122 123 124 125 126 127 128 129 131 132 134 135
 136 137 139 140 141 142 143 144 145 146 147 149 150 151 152 154 155 156
 157 158 159 161 163 164 167 169 170 171 172 174 175 176 177 178 180 181
 182 184 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 202
 204 207 208 209 211 212 213 216 217 218 219 220 222 223 224 225 227 228
 230 232 234 235 236 237 238 239 241 242 243 244 245 246 247 248 250 251
 252 254 256 257 258 259 261 262 263 265 267 268 269 270 271 272 273 274
 275 276 277 278 279 281 282 284 285 286 287 288 289 290 292 293 294 295
 296 297 298 299 300 302 303 304 305 306 307 308 309 311 313 314 315 316
 317 319 320 321 323 324 325 326 327 330 331 332 333 334 335 336 337 338
 339 340 341 342 343 344 345 346 347 348 350 351 352 353 354 355 356 359
 360 361 362 363 364 365 367 368 369 370 371 372 374 375 376 377 378 379
 380 383 384 385 386 387 389 390 391 392 393 394 395 396 397 398 399 400
 401 402 403 404 405 406 408 410 412 413 414 415 416 417 418 419 420 421
 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440
 441 442 443 444 448 449 450 451 452 453 454 455 456 457 458 459 460 461
 462 463 464 466 467 470 471 472 473 474 475 476 477 478 479 481 482 484
 487 488 490 491 492 493 495 497 498 499 500 503 504 505 506 507 508 509
 511 512 515 516 517 518 519 520 521 522 523 524 525 528 529 530 532 533
 535 536 538 540 541 546 547 548 549 550 552 553 554 555 556 557 559 560
 561 562 563 565 566 567 568 570 571 572 573 574 575 579 580 583 584 585
 586 587 588 589 591 592 593 595 596 597 598 599 600 602 603 604 605 606
 608 609 610 611 613 614 616 617 618 619 620 621 622 623 626 627 629 630
 631 634 636 638 639 640 641 642 643 644 645 646 647 650 652 653 654 655
 656 657 659 660 661 663 664 668 669 670 671 672 673 674 675 676 679 680
 681 682 683 684 685 687 688 689 690 691 693 694 696 697 698 700 701 702
 703 705 706 707 708 709 710 711 712 713 715 716 717 718 719 722 723 724
 725 726 727 728 729 730 732 733 734 735 736 737 739 740 741 742 743 744
 745 746 747 748 749 750 751 752 754 756 757 758 759 760 761 762 763], TEST: [  0   2   4   5  11  14  18  21  28  30  32  3
  65  74  93  94  96  97  98 121 130 133 138 148 153 160 162 165 166 168
```

```
 173 179 183 185 201 203 205 206 210 214 215 221 226 229 231 233 240 249
 253 255 260 264 266 280 283 291 301 310 312 318 322 328 329 349 357 358
 366 373 381 382 388 407 409 411 422 445 446 447 465 468 469 480 483 485
 486 489 494 496 501 502 510 513 514 526 527 531 534 537 539 542 543 544
 545 551 558 564 569 576 577 578 581 582 590 594 601 607 612 615 624 625
 628 632 633 635 637 648 649 651 658 662 665 666 667 677 678 686 692 695
 699 704 714 720 721 731 738 753 755]
TRAIN: [  0   2   3   4   5   6   7   9  10  11  12  13  14  15  16  18  19  20
  21  22  23  24  26  27  28  29  30  31  32  33  34  35  37  39  41  42
  43  44  45  46  47  48  49  51  52  54  55  56  57  58  59  61  62  63
  64  65  66  68  69  70  71  72  73  74  75  76  79  80  81  83  85  86
  87  88  90  92  93  94  95  96  97  98  99 101 104 106 108 109 110 111
 113 115 116 118 119 120 121 123 124 125 126 127 128 129 130 132 133 134
 137 138 139 140 141 142 143 144 145 148 149 150 151 153 155 156 158 159
 160 161 162 163 164 165 166 167 168 169 170 171 173 177 179 180 183 184
 185 187 189 190 192 195 196 197 198 199 201 202 203 204 205 206 208 210
 211 212 214 215 216 217 218 220 221 224 225 226 227 228 229 230 231 232
 233 234 235 237 238 239 240 241 243 244 245 246 247 248 249 250 251 252
 253 254 255 256 257 258 260 261 263 264 265 266 267 270 272 273 274 276
 277 278 279 280 281 282 283 286 287 288 289 291 292 293 294 295 296 297
 298 299 300 301 302 303 304 305 306 307 309 310 311 312 313 314 315 316
```

```
svn=SVC()
```

```
print((train_index.shape[0], test_index.shape[0]))
```

```
(612, 152)
```

```
X_train, X_test = X[train_index], X[test_index]
y_train, y_test = Y[train_index], Y[test_index]
```

```
b=svn.fit(X_train,y_train)
```

```
p=b.predict(X_test)
```

```
#SVC accuracy KFold
accuracy=accuracy_score(y_true=y_test, y_pred=p)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 48.03%
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf=RandomForestClassifier()
```

```
c=rf.fit(X_train,y_train)
```

```
p=c.predict(X_test)
```

```
#RF accuracy KFold
accuracy=accuracy_score(y_true=y_test, y_pred=p)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 76.32%
```

```
from sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
```

```
e=classifier.fit(X_train, y_train)
```

```
prediction= e.predict(X_test)
```

```
#DTC accuracy KFold
accuracy=accuracy_score(y_true=y_test, y_pred=prediction)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 55.26%
```

```
#StratifiedKFold
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=37)

for train_index, test_index in skf.split(X, Y):
    print(f'TRAIN: {train_index}, TEST: {test_index}')
```

```
TRAIN: [  0   1   4   6   7   8   9  11  12  13  14  15  17  18  19  20  21  22
  23  25  27  28  29  30  31  33  34  35  36  37  38  39  40  41  42  43
  44  45  46  48  49  50  51  52  54  55  56  57  58  59  60  61  62  63
  64  65  67  69  70  72  73  74  75  76  77  78  79  81  82  83  84  85
  87  89  90  91  92  93  95  96  97  98  99 100 101 104 105 106 107 112
 113 114 115 117 118 119 120 121 122 123 125 126 127 128 129 131 132 133
 134 135 136 138 139 140 141 142 143 146 148 149 150 151 152 155 157 158
 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178
 179 182 183 184 185 187 188 190 191 192 193 194 196 199 201 202 203 204
 205 206 207 208 209 211 212 214 217 218 219 220 221 223 224 227 228 230
 231 232 233 234 235 237 239 240 242 243 244 245 247 248 250 251 252 253
 254 255 258 260 261 262 263 264 266 268 271 272 273 274 275 276 278 279
 280 281 282 283 286 287 288 289 290 291 292 293 295 296 299 300 301 302
 303 304 305 306 307 308 309 310 311 312 313 314 316 317 318 319 320 322
 324 326 328 329 330 331 332 333 334 335 336 337 340 341 342 343 344 345
 346 348 349 351 352 353 354 355 356 357 358 360 361 362 363 365 366 367
 368 369 370 371 372 373 374 375 376 378 379 380 381 386 388 389 390 391
 392 393 396 397 398 399 401 402 403 404 405 406 407 408 409 410 411 412
 413 414 415 416 417 418 421 422 423 425 426 427 429 430 431 433 434 437
 438 439 440 441 442 443 444 445 446 448 449 450 451 452 453 454 455 457
 458 459 460 461 462 463 464 466 467 468 470 471 472 473 475 476 477 478
 479 481 482 483 485 486 487 488 489 490 491 492 493 495 497 498 499 500
 501 503 504 505 508 509 510 511 512 513 514 515 516 517 519 520 521 522
 523 525 526 527 528 529 530 531 533 534 535 536 537 538 539 540 541 542
 543 544 545 546 548 549 551 552 553 554 555 556 557 558 559 560 561 563
 564 565 566 567 568 571 572 573 574 575 576 577 579 580 581 584 585 586
 587 590 593 594 596 598 602 603 604 605 606 608 609 611 612 613 615 616
 617 618 619 621 622 623 624 625 626 627 628 629 632 633 634 635 636 637
 638 639 641 642 643 644 647 648 649 650 651 652 653 654 655 656 659 660
 661 662 663 664 665 667 668 669 670 672 673 674 675 678 679 681 682 683
 684 685 686 687 688 689 690 691 693 695 696 697 698 699 700 701 702 703
 705 706 707 708 709 711 712 713 714 715 716 717 718 719 720 721 722 723
 724 725 726 727 728 729 730 732 733 734 736 737 738 739 740 741 742 743
 744 745 746 748 749 750 751 752 753 754 755 756 757 760 761 762 763], TEST: [  2   3   5  10  16  24  26  32  47  53  66  6
 103 108 109 110 111 116 124 130 137 144 145 147 153 154 156 159 160 180
 181 186 189 195 197 198 200 210 213 215 216 222 225 226 229 236 238 241
 246 249 256 257 259 265 267 269 270 277 284 285 294 297 298 315 321 323
 325 327 338 339 347 350 359 364 377 382 383 384 385 387 394 395 400 419
 420 424 428 432 435 436 447 456 465 469 474 480 484 494 496 502 506 507
 518 524 532 547 550 562 569 570 578 582 583 588 589 591 592 595 597 599
 600 601 607 610 614 620 630 631 640 645 646 657 658 666 671 676 677 680
 692 694 704 710 731 735 747 758 759]
TRAIN: [  1   2   3   4   5   7   8  10  11  12  13  15  16  17  18  19  20  21
  23  24  25  26  27  28  29  30  31  32  33  35  36  37  38  40  41  42
  43  45  46  47  49  50  51  52  53  55  56  57  59  60  61  62  63  65
  66  67  68  69  71  72  73  74  76  77  78  80  81  82  85  86  87  88
  89  90  94  95  96  97  98  99 100 101 102 103 104 106 107 108 109 110
 111 112 113 114 115 116 117 118 119 120 121 123 124 125 126 127 128 129
 130 131 132 134 135 136 137 138 139 140 142 143 144 145 147 148 149 150
 151 152 153 154 155 156 157 158 159 160 161 163 165 166 167 168 170 172
 173 175 176 177 179 180 181 182 183 184 186 187 189 190 191 192 193 194
 195 196 197 198 200 201 202 203 204 205 206 208 210 211 212 213 214 215
 216 218 219 220 221 222 223 225 226 227 228 229 230 231 234 236 237 238
 239 240 241 244 246 249 250 251 253 254 255 256 257 259 260 262 263 265
 266 267 268 269 270 272 273 274 275 277 281 282 283 284 285 286 287 288
 289 291 292 293 294 295 296 297 298 299 302 303 304 305 306 309 310 311
 312 314 315 316 318 319 320 321 322 323 324 325 326 327 328 329 330 331
```

```
X_train, X_test = X[train_index], X[test_index]
y_train, y_test = Y[train_index], Y[test_index]


f=svn.fit(X_train,y_train)


p=f.predict(X_test)


#SVC accuracy StratifiedKFold
accuracy=accuracy_score(y_true=y_test, y_pred=p)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 46.05%
```

```
from sklearn.ensemble import RandomForestClassifier


rf=RandomForestClassifier()
```

```
g=rf.fit(X_train,y_train)


p=g.predict(X_test)


#RF accuracy StratifiedKFold
accuracy=accuracy_score(y_true=y_test, y_pred=p)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 79.61%
```

```
from sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)


x=classifier.fit(X_train,y_train)


p=x.predict(X_test)


accuracy=accuracy_score(y_true=y_test, y_pred=p)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 64.47%
```

```
#ShuffleSplit_MonteCarlo
from sklearn.model_selection import ShuffleSplit

ss = ShuffleSplit(n_splits=6, random_state=37)

for train_index, test_index in ss.split(X, Y):
    print(f'TRAIN: {train_index}, TEST: {test_index}')
```

```
TRAIN: [666 162 577 233 215 168 183 612 738  74 628 558 551 496   2 206 607 255
 534 537 695 422 231 310   5 160  11  93 731 153 501 632 266 138 721 665
  30 166 539 544 382 543  34 578 526  21 686 755 469 148  96 704  98 226
 446 486 214 357 349 678 564 662 483 485 407 447 531 513 468 480 692 502
 185 291 582 494 152 341   8 754 500 334 549 574 760  78 209 191 484 262
  60 521 117  38 368 563 580 269 268 102 530  53 284 131 397  36 538  17
 751 722 684 386 200 188 756 689 470 718 147  77  67 383 492 242 572 618
 745 271  82 186 408 174 733 708 495 420 369 758   1 259 713 175 723 389
 172 176 703 154 395 588 146 223 290 105 680 178  50 136 621 421  89 698
 364 707 541 706 417 622 676 122 525 462 597 430 606 436 112 509 647 636
 193 476 157 449 736 522 107 194 103  40 552 337 207  84 616 763 342 135
 653 556 735 100 646 275 213 236 325  91 592 285 338 681 659 308 219 473
 182 339 516 518 222  25 579 608 181 114 700 443 320 258 140 126 491 111
 252 410 697 548 600 141 298 432 669 626 276  16 155 158 180 442 171 304
  72  24 286 288 613 503   6 583 118 761  43 110 293 303 475 263 235 694
 208 623  20 702 187 460 591 134 467 406 125 161 663 507 150 327 424 109
 232  76 734 393 101 197  75 472 265  10 528 687 481 533 440  92 673 471
 132 354 379  52 401 748 644 701 431  54 672 726 346 585  42 660 204 392
 400 752 605 143 151 353 498 323 198 550 656 747 493 279  27 243 113 324
 372   3 438 169 362 335 315 441 466 177 559 238 617 586 257 448 433 742
 750 224 593 403 450 560  66 643 567 330  70 245 163  71 261 294 740  61
 123 724 685 645 384 690 630 319  39 196 573 705 650  26 331 212 332 737
 343 487 490 359 360 746 657 453 247 129 311 732 416 348 307 679 553  80
  86 739 120 555  55 595 614 274 314 218 642 515 524  68 429 693  95 313
 376 277 405 227 237 216 124 655 627 380 256 340 455 620 350  99 371 396
 351 297  81 479 463  47 278  88 710 144 529 719  90 217 267 211 604 532
 302 164 139 674 296 520 374 584 377 347 517 282 629 104 506 390 418 127
 378 272 725 557 289 241 535 457 640 683  31 639 728 654 729 461 682 610
 248 402 425 156 499  44 464 743 195 547  83 190 352 691  12 239 741 458
 715 609 108  23 712 474 142 326 270 145 230 759 428  69 571 603 394 668
 587 370  13 300 356 170 546 220  58 184 670  45 762 508 427 385 299 254
 711 589 511 305 598 688  64 536 456 512 106 399  85 505 661   9 375 287
 419 137 413 565  33 478 199 306 295  29 596  41  73 709  79 189 631 423
 523 459  51 246 414 619  57   7 730 561 671 333  87 361 228 292 281 444
 599 225 367 452 316 192 716 426 749 568 119 167 336 391 321 504 398 696
 317 250 439 128 415 497 115 454 435 149 477 566 717 116 744 365 540 562
  15  35 273 404 234 641 309 519 638 159 570 757 345 482 355 412 611 202
 244 664  19 602 652  46 434 387 488  62 727 344 575 554 451 675 634  22
 437 251 363], TEST: [409  32 601 590  97 283 625 489 637   4 381   0 411  37 527  59 545 366
 264 328 322 133 615 121 165 173 388  28  63 649 210 667 205 699 635 130
 648 301 465 240 542  18 445 576 280  65 658 677 201 720 633 714  49 581
 229 514 651 329 318  56 312 569 753  48 249 594 203 179 358 260 373  94
  14 253 624 510 221]
TRAIN: [606 707 251 343 718 292 257 744 680 665 436 686 174 322 352 459 458 168
 607 622 208 535 498 148  96 334 685 217 369 183 440  47 164 538 529 244
 140   5 611 348 372 542 496 617 480 238 141  33  54  73 401 151 732 657
   3 463 342 214 761 206  83 108  66 180 267 184 523 249 395 354 335 196
 407 725  55 313 364 582 672 190   8 752 123 557  45 628  44 158 475 157
 754 409 167  80 598 172 635 534 658 243 353 411 199 449 273  60 493 676
  13 179  23 572 638 547 143 577 531 294 144 675 112 550 711 444  38 387
 381 181   2 286  11 327 315 651 567 104  87 483 404 583  40 434  74 532
```

```
      631 155 227 476 518 479 579 308 466 749 720 484 737 560 115 161 712 166
      311 624 673 225   0 136 690 132 420 613 209 207 258 708 481 290  41  81
      403 588  68 149 464 753 252  98 330 195 491  49  63 316 661  31 591 746
      691  92 279 325 374 413 389 527  95 117 153 478 376 594 416 255 100 370
      649 759 103 573 138 162 295 296 318 378 747  67  90 467 569 521 220 323
      422 246 650 693 368 525 612 360 625 259 637 129 548 454 397   6 570 234
      456 503 492 105 349 701 629 554 627 437 654 723 558 122 666 757  14 677
```

```
X_train, X_test = X[train_index], X[test_index]
y_train, y_test = Y[train_index], Y[test_index]
```

```
t=classifier.fit(X_train,y_train)
```

```
p=t.predict(X_test)
```

```
accuracy=accuracy_score(y_true=y_test, y_pred=p)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 63.64%
```

```
h=svn.fit(X_train,y_train)
```

```
p=h.predict(X_test)
```

```
#SVC accuracy ShuffleSplit
accuracy=accuracy_score(y_true=y_test, y_pred=p)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 53.25%
```

```
from sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
```

```
v=classifier.fit(X_train,y_train)
```

```
p=v.predict(X_test)
```

```
accuracy=accuracy_score(y_true=y_test, y_pred=p)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 63.64%
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf=RandomForestClassifier()
```

```
i=rf.fit(X_train,y_train)
```

```
p=i.predict(X_test)
```

```
#RF accuracy ShuffleSplit
accuracy=accuracy_score(y_true=y_test, y_pred=p)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 84.42%
```

```
import pickle as cPickle
with open('/content/drive/MyDrive/F_Genre.h5', 'wb') as f:
    cPickle.dump(i, f)
```

```
#StratifiedShuffleSplit
from sklearn.model_selection import StratifiedShuffleSplit

sss = StratifiedShuffleSplit(n_splits=6, random_state=37)

for train_index, test_index in sss.split(X, Y):
    print(f'TRAIN: {train_index}, TEST: {test_index}')
```

```
TRAIN: [ 90 298 702 684  84  13 662 315  54 429 541 643   6 568 498 158 209 165
  380 580  31 624 330 239 729 171 660 747 689 127 316 670 619 227 752 563
  531  28 143 730 486 321 177 625 322 510 157 739 495 291 139  72 518 288
  317  79 602 712 186 478 263 542 549 258 483 135 519 644 122 118 236 569
```

```
      615 691 575   3 404 350 249 212 672 603 692 327  16 345 755 481  62 383
      479 168 182 714 754 170   4 314 410 361 577 409 193 223 722   7 306 323
      705 373 233 652 592 217 463 586 156 366 253 148 108 673 213 105 235 724
      325 375 725 532  38 267 140 358 376 174  52 558 417 398  95 290  80 229
      419 251  75 554   8 226 134 241 653 613 107 102 449 328 741 514 399 508
      504 309 178 282 368 183 493 234 273 520 382 470 701 195 641 230 301  27
      460 406 506  34 589 276 124 199 438 132 537 489 761   0 694 194 581 680
      331 248 100 681 646 614 257 400 401 664 261 166 516  58  48 686 391  49
      392 203 145  85 721 497  94  20 303 703 649 567 210 445 297 490 200  35
      668 693  57 299  82  39 605 152 207 167 370 285 245 661 667 180 304 753
      634 512 601 110  15 606 364 281 206  98  36 435 162 450 591 659 695 341
      501  64 150  89 416 630 279 607 707 224  51 582 439 723 559 240 742 231
       17 491 161 120 343  23 727 610 172 612 216 757 671 151 287 574 126 125
      632 423  40 379 455 260 286 377  29 631 337 633 431 487 709 456 482 485
      584 176 678 697 620 494 617 259 252 360 552 738 356 403 719 142  47 305
      117 243  66 534 385 759 576 562 578 704 525 539  83 396 528 289 332 595
      687  37 232 440 326  10 198 736 596 647 544 131 160  46 650 597 645 708
      111 205 593 716 710 103 484 146  76 740 763 188 529 412 750  70 141 393
       24 270 238  61 732 201  30 458 731 144 202  88 452 437 121 427 367  92
       63 196 215 269 536 533 112 335 274 683 153  12  26  53 184 136 250 448
      242 220  41  59 720 640 526 275 500  87 149 690 745 480 268 244 443  56
      608 302 106 527 278 265 748 611 557 651 130  81 218 760 421  19 476 590
      565 587 746 272 553  97 191 464  50 292 428 473 743 465 699  78 296 616
      413 424 454 155 524 189 247 277 749 225 333 163 308  67 509 762 432 505
      415 185   1 109 362 638  25  18 372 451 627 560 564 442 329 535 511  60
      618 256 407 675 334 271  99 354 119 471 585 280 340 204 475 503 418 324
       96 386 677 735 666 388 214  73 433 387 246 338 444 138 756 447 221  22
      320 715 628 430 353 642 192 346  43 657 556  91 457 674 551  71 197 547
      550 425 656 540 129   9 179 496 713 397 262 648 389 228 700 264 588 266
      548  11 573 357 734 222 349 411 294 115 682 637 515 133 717 318 351 319
      629 441   5 635  55 384 676 604 104 113 395 545 453 513 698 696 446 123
      599  14 347  45 570 655 374 517  86 336 283 718 566  33 128 706 169 414
      669 307 293 621 284   2 422 737 339 502 462 175 154 101 468 187 543  69
      219 546 420  21 434 523 530 469 159 352 381 636 499 665 147 116 378 609
      211 459 390], TEST: [477 402 572 639 359 363 623 600  65 436 598 365 711 751 369 344 114 726
      488 426 164 237 654 371 408 733 555 254 728 173 311 467 522  42 312 571
      137 626 685 688 313 583 521 663 466 579 208 507 255 310 461  32 474 744
      181 679 348 405 622 472  74  93  68 594 561 300 658 295 355 190 342  44
      758  77 394 538 492]
     TRAIN: [653 509 406 364 184 730 166 742  28 241 263 331 490 673 593 723 541 195
       98 552 676 235 461  64  12  40 505 427 416 156 754  54 257 125 168 308
      688 689 762  47 687 107 320 139 631 679 751 662 696 332 592 371 143 544
      170 652  70 430 285  94  57 647 618 680  62  44 163 480 133 646 267 227
      158 556 254 640  14 603 258 659 597 750 293 629 117 657 701 401 100 216
      743 577 108  89 307 608 515 325 622 239 288  27  97  42 700 146 732 268
      103 569 149 169 496  51 436 614 528 488 747 347  45 192 213 693 376  48
       59 690 694 144 664 398 705 615 198 472 417 294 335  61 400 610  95 632
       32 102 601  67 181 413 344 532 620 584 605 114  83 250 366 261 207 477
      483 141 387 561 621 491 217 468 230 644 539 287 129 512 224 649 361 302
      363 529 442 208 240 220 377 419 563 403 137  13 489  63  58 521  56 585
       18 511 348 642 439  22 695 385 350 111 526 699 486 185 669 339 232  93
      360 663 395 292 535 145 709 744 289 330 598 178 504 763 297 380 365 514
       46 660 717 253 570 656  29 189 352 283 362 155 713 576 369 259 446 126
```

```
X_train, X_test = X[train_index], X[test_index]
y_train, y_test = Y[train_index], Y[test_index]
```

```
k=classifier.fit(X_train,y_train)
```

```
p=k.predict(X_test)
```

```
accuracy=accuracy_score(y_true=y_test, y_pred=p)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 49.35%
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf=RandomForestClassifier()
```

```
d=rf.fit(X_train,y_train)
```

```
p=d.predict(X_test)
```

```
#RF accuracy StratifiedShuffleSplit
accuracy=accuracy_score(y_true=y_test, y_pred=p)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 67.53%
```

```
import joblib
```

```
filename = 'music_genre_f.pkl'
joblib.dump(d, filename)
```

```
['music_genre_f.pkl']
```

```
import pickle as cPickle
with open('/content/drive/MyDrive/A_Genre.h5', 'wb') as f:
    cPickle.dump(d, f)
```

```
import pickle as cPickle
with open('/content/drive/MyDrive/B_Genre.h5', 'wb') as f:
    cPickle.dump(a, f)
```