

This is data analysis and model fit for human activity. Data is collected using smartphone sensors.

Data analysis

```
In [1]: 1 import numpy as np
2 import pandas as pd
3 import math
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6 import psutil
7 import time
8
9 from sklearn.preprocessing import StandardScaler
10 from sklearn.preprocessing import normalize
11 from sklearn.cluster import DBSCAN
12 from sklearn import metrics
13 from sklearn.metrics import silhouette_score
14 from itertools import product
15 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
16 from sklearn.neighbors import NearestNeighbors
17
18 #Setting the plot parameters for standardized graphs.
19 plt.rcParams["figure.figsize"] = (15, 7)
20 plt.style.use("ggplot")
```

```
In [2]: 1 #Read the data. 'train_data' includes features and label.
2 train_data = pd.read_csv("Dataset.csv") #Feature+Labels.
3 X = np.array(pd.read_csv("X_train.csv", header=None)) #Only features.
4 label = pd.read_csv("y_train.csv", header=None)
```

```
In [ ]: 1 #Scatter plot of two features to see how data is spread. Data looks very dense and overlapping
2 sns.scatterplot(X[:,2], X[:,3], hue=train_data['Activity'], palette='bright')
```

```
In [ ]: 1 label.shape
```

```
In [ ]: 1 #Reference key for Labels.
2 KeyRef = {1:"WALKING", 2:"WALKING_UPSTAIRS", 3:"WALKING_DOWNSTAIRS", 4:"SITTING", 5:"STANDING", 6:"LAYING", 7:"STANDING_UPSTAIRS", 8:"STANDING_DOWNSTAIRS", 9:"WALKING_UPSTAIRS", 10:"WALKING_DOWNSTAIRS", 11:"SITTING_UPSTAIRS", 12:"SITTING_DOWNSTAIRS", 13:"Lying_Side", 14:"Lying_Front", 15:"Lying_Belly", 16:"STANDING_UPSTAIRS", 17:"STANDING_DOWNSTAIRS", 18:"SITTING_UPSTAIRS", 19:"SITTING_DOWNSTAIRS", 20:"Lying_Side", 21:"Lying_Front", 22:"Lying_Belly", 23:"WALKING_UPSTAIRS", 24:"WALKING_DOWNSTAIRS", 25:"SITTING_UPSTAIRS", 26:"SITTING_DOWNSTAIRS", 27:"Lying_Side", 28:"Lying_Front", 29:"Lying_Belly", 30:"STANDING_UPSTAIRS", 31:"STANDING_DOWNSTAIRS", 32:"SITTING_UPSTAIRS", 33:"SITTING_DOWNSTAIRS", 34:"Lying_Side", 35:"Lying_Front", 36:"Lying_Belly", 37:"WALKING_UPSTAIRS", 38:"WALKING_DOWNSTAIRS", 39:"SITTING_UPSTAIRS", 40:"SITTING_DOWNSTAIRS", 41:"Lying_Side", 42:"Lying_Front", 43:"Lying_Belly", 44:"STANDING_UPSTAIRS", 45:"STANDING_DOWNSTAIRS", 46:"SITTING_UPSTAIRS", 47:"SITTING_DOWNSTAIRS", 48:"Lying_Side", 49:"Lying_Front", 50:"Lying_Belly", 51:"WALKING_UPSTAIRS", 52:"WALKING_DOWNSTAIRS", 53:"SITTING_UPSTAIRS", 54:"SITTING_DOWNSTAIRS", 55:"Lying_Side", 56:"Lying_Front", 57:"Lying_Belly", 58:"STANDING_UPSTAIRS", 59:"STANDING_DOWNSTAIRS", 60:"SITTING_UPSTAIRS", 61:"SITTING_DOWNSTAIRS", 62:"Lying_Side", 63:"Lying_Front", 64:"Lying_Belly", 65:"WALKING_UPSTAIRS", 66:"WALKING_DOWNSTAIRS", 67:"SITTING_UPSTAIRS", 68:"SITTING_DOWNSTAIRS", 69:"Lying_Side", 70:"Lying_Front", 71:"Lying_Belly", 72:"STANDING_UPSTAIRS", 73:"STANDING_DOWNSTAIRS", 74:"SITTING_UPSTAIRS", 75:"SITTING_DOWNSTAIRS", 76:"Lying_Side", 77:"Lying_Front", 78:"Lying_Belly", 79:"WALKING_UPSTAIRS", 80:"WALKING_DOWNSTAIRS", 81:"SITTING_UPSTAIRS", 82:"SITTING_DOWNSTAIRS", 83:"Lying_Side", 84:"Lying_Front", 85:"Lying_Belly", 86:"STANDING_UPSTAIRS", 87:"STANDING_DOWNSTAIRS", 88:"SITTING_UPSTAIRS", 89:"SITTING_DOWNSTAIRS", 90:"Lying_Side", 91:"Lying_Front", 92:"Lying_Belly", 93:"WALKING_UPSTAIRS", 94:"WALKING_DOWNSTAIRS", 95:"SITTING_UPSTAIRS", 96:"SITTING_DOWNSTAIRS", 97:"Lying_Side", 98:"Lying_Front", 99:"Lying_Belly", 100:"STANDING_UPSTAIRS", 101:"STANDING_DOWNSTAIRS", 102:"SITTING_UPSTAIRS", 103:"SITTING_DOWNSTAIRS", 104:"Lying_Side", 105:"Lying_Front", 106:"Lying_Belly", 107:"WALKING_UPSTAIRS", 108:"WALKING_DOWNSTAIRS", 109:"SITTING_UPSTAIRS", 110:"SITTING_DOWNSTAIRS", 111:"Lying_Side", 112:"Lying_Front", 113:"Lying_Belly", 114:"STANDING_UPSTAIRS", 115:"STANDING_DOWNSTAIRS", 116:"SITTING_UPSTAIRS", 117:"SITTING_DOWNSTAIRS", 118:"Lying_Side", 119:"Lying_Front", 120:"Lying_Belly", 121:"WALKING_UPSTAIRS", 122:"WALKING_DOWNSTAIRS", 123:"SITTING_UPSTAIRS", 124:"SITTING_DOWNSTAIRS", 125:"Lying_Side", 126:"Lying_Front", 127:"Lying_Belly", 128:"STANDING_UPSTAIRS", 129:"STANDING_DOWNSTAIRS", 130:"SITTING_UPSTAIRS", 131:"SITTING_DOWNSTAIRS", 132:"Lying_Side", 133:"Lying_Front", 134:"Lying_Belly", 135:"WALKING_UPSTAIRS", 136:"WALKING_DOWNSTAIRS", 137:"SITTING_UPSTAIRS", 138:"SITTING_DOWNSTAIRS", 139:"Lying_Side", 140:"Lying_Front", 141:"Lying_Belly", 142:"STANDING_UPSTAIRS", 143:"STANDING_DOWNSTAIRS", 144:"SITTING_UPSTAIRS", 145:"SITTING_DOWNSTAIRS", 146:"Lying_Side", 147:"Lying_Front", 148:"Lying_Belly", 149:"WALKING_UPSTAIRS", 150:"WALKING_DOWNSTAIRS", 151:"SITTING_UPSTAIRS", 152:"SITTING_DOWNSTAIRS", 153:"Lying_Side", 154:"Lying_Front", 155:"Lying_Belly", 156:"STANDING_UPSTAIRS", 157:"STANDING_DOWNSTAIRS", 158:"SITTING_UPSTAIRS", 159:"SITTING_DOWNSTAIRS", 160:"Lying_Side", 161:"Lying_Front", 162:"Lying_Belly", 163:"WALKING_UPSTAIRS", 164:"WALKING_DOWNSTAIRS", 165:"SITTING_UPSTAIRS", 166:"SITTING_DOWNSTAIRS", 167:"Lying_Side", 168:"Lying_Front", 169:"Lying_Belly", 170:"STANDING_UPSTAIRS", 171:"STANDING_DOWNSTAIRS", 172:"SITTING_UPSTAIRS", 173:"SITTING_DOWNSTAIRS", 174:"Lying_Side", 175:"Lying_Front", 176:"Lying_Belly", 177:"WALKING_UPSTAIRS", 178:"WALKING_DOWNSTAIRS", 179:"SITTING_UPSTAIRS", 180:"SITTING_DOWNSTAIRS", 181:"Lying_Side", 182:"Lying_Front", 183:"Lying_Belly", 184:"STANDING_UPSTAIRS", 185:"STANDING_DOWNSTAIRS", 186:"SITTING_UPSTAIRS", 187:"SITTING_DOWNSTAIRS", 188:"Lying_Side", 189:"Lying_Front", 190:"Lying_Belly", 191:"WALKING_UPSTAIRS", 192:"WALKING_DOWNSTAIRS", 193:"SITTING_UPSTAIRS", 194:"SITTING_DOWNSTAIRS", 195:"Lying_Side", 196:"Lying_Front", 197:"Lying_Belly", 198:"STANDING_UPSTAIRS", 199:"STANDING_DOWNSTAIRS", 200:"SITTING_UPSTAIRS", 201:"SITTING_DOWNSTAIRS", 202:"Lying_Side", 203:"Lying_Front", 204:"Lying_Belly", 205:"WALKING_UPSTAIRS", 206:"WALKING_DOWNSTAIRS", 207:"SITTING_UPSTAIRS", 208:"SITTING_DOWNSTAIRS", 209:"Lying_Side", 210:"Lying_Front", 211:"Lying_Belly", 212:"STANDING_UPSTAIRS", 213:"STANDING_DOWNSTAIRS", 214:"SITTING_UPSTAIRS", 215:"SITTING_DOWNSTAIRS", 216:"Lying_Side", 217:"Lying_Front", 218:"Lying_Belly", 219:"WALKING_UPSTAIRS", 220:"WALKING_DOWNSTAIRS", 221:"SITTING_UPSTAIRS", 222:"SITTING_DOWNSTAIRS", 223:"Lying_Side", 224:"Lying_Front", 225:"Lying_Belly", 226:"STANDING_UPSTAIRS", 227:"STANDING_DOWNSTAIRS", 228:"SITTING_UPSTAIRS", 229:"SITTING_DOWNSTAIRS", 230:"Lying_Side", 231:"Lying_Front", 232:"Lying_Belly", 233:"WALKING_UPSTAIRS", 234:"WALKING_DOWNSTAIRS", 235:"SITTING_UPSTAIRS", 236:"SITTING_DOWNSTAIRS", 237:"Lying_Side", 238:"Lying_Front", 239:"Lying_Belly", 240:"STANDING_UPSTAIRS", 241:"STANDING_DOWNSTAIRS", 242:"SITTING_UPSTAIRS", 243:"SITTING_DOWNSTAIRS", 244:"Lying_Side", 245:"Lying_Front", 246:"Lying_Belly", 247:"WALKING_UPSTAIRS", 248:"WALKING_DOWNSTAIRS", 249:"SITTING_UPSTAIRS", 250:"SITTING_DOWNSTAIRS", 251:"Lying_Side", 252:"Lying_Front", 253:"Lying_Belly", 254:"STANDING_UPSTAIRS", 255:"STANDING_DOWNSTAIRS", 256:"SITTING_UPSTAIRS", 257:"SITTING_DOWNSTAIRS", 258:"Lying_Side", 259:"Lying_Front", 260:"Lying_Belly", 261:"WALKING_UPSTAIRS", 262:"WALKING_DOWNSTAIRS", 263:"SITTING_UPSTAIRS", 264:"SITTING_DOWNSTAIRS", 265:"Lying_Side", 266:"Lying_Front", 267:"Lying_Belly", 268:"STANDING_UPSTAIRS", 269:"STANDING_DOWNSTAIRS", 270:"SITTING_UPSTAIRS", 271:"SITTING_DOWNSTAIRS", 272:"Lying_Side", 273:"Lying_Front", 274:"Lying_Belly", 275:"WALKING_UPSTAIRS", 276:"WALKING_DOWNSTAIRS", 277:"SITTING_UPSTAIRS", 278:"SITTING_DOWNSTAIRS", 279:"Lying_Side", 280:"Lying_Front", 281:"Lying_Belly", 282:"STANDING_UPSTAIRS", 283:"STANDING_DOWNSTAIRS", 284:"SITTING_UPSTAIRS", 285:"SITTING_DOWNSTAIRS", 286:"Lying_Side", 287:"Lying_Front", 288:"Lying_Belly", 289:"WALKING_UPSTAIRS", 290:"WALKING_DOWNSTAIRS", 291:"SITTING_UPSTAIRS", 292:"SITTING_DOWNSTAIRS", 293:"Lying_Side", 294:"Lying_Front", 295:"Lying_Belly", 296:"STANDING_UPSTAIRS", 297:"STANDING_DOWNSTAIRS", 298:"SITTING_UPSTAIRS", 299:"SITTING_DOWNSTAIRS", 300:"Lying_Side", 301:"Lying_Front", 302:"Lying_Belly", 303:"WALKING_UPSTAIRS", 304:"WALKING_DOWNSTAIRS", 305:"SITTING_UPSTAIRS", 306:"SITTING_DOWNSTAIRS", 307:"Lying_Side", 308:"Lying_Front", 309:"Lying_Belly", 310:"STANDING_UPSTAIRS", 311:"STANDING_DOWNSTAIRS", 312:"SITTING_UPSTAIRS", 313:"SITTING_DOWNSTAIRS", 314:"Lying_Side", 315:"Lying_Front", 316:"Lying_Belly", 317:"WALKING_UPSTAIRS", 318:"WALKING_DOWNSTAIRS", 319:"SITTING_UPSTAIRS", 320:"SITTING_DOWNSTAIRS", 321:"Lying_Side", 322:"Lying_Front", 323:"Lying_Belly", 324:"STANDING_UPSTAIRS", 325:"STANDING_DOWNSTAIRS", 326:"SITTING_UPSTAIRS", 327:"SITTING_DOWNSTAIRS", 328:"Lying_Side", 329:"Lying_Front", 330:"Lying_Belly", 331:"WALKING_UPSTAIRS", 332:"WALKING_DOWNSTAIRS", 333:"SITTING_UPSTAIRS", 334:"SITTING_DOWNSTAIRS", 335:"Lying_Side", 336:"Lying_Front", 337:"Lying_Belly", 338:"STANDING_UPSTAIRS", 339:"STANDING_DOWNSTAIRS", 340:"SITTING_UPSTAIRS", 341:"SITTING_DOWNSTAIRS", 342:"Lying_Side", 343:"Lying_Front", 344:"Lying_Belly", 345:"WALKING_UPSTAIRS", 346:"WALKING_DOWNSTAIRS", 347:"SITTING_UPSTAIRS", 348:"SITTING_DOWNSTAIRS", 349:"Lying_Side", 350:"Lying_Front", 351:"Lying_Belly", 352:"STANDING_UPSTAIRS", 353:"STANDING_DOWNSTAIRS", 354:"SITTING_UPSTAIRS", 355:"SITTING_DOWNSTAIRS", 356:"Lying_Side", 357:"Lying_Front", 358:"Lying_Belly", 359:"WALKING_UPSTAIRS", 360:"WALKING_DOWNSTAIRS", 361:"SITTING_UPSTAIRS", 362:"SITTING_DOWNSTAIRS", 363:"Lying_Side", 364:"Lying_Front", 365:"Lying_Belly", 366:"STANDING_UPSTAIRS", 367:"STANDING_DOWNSTAIRS", 368:"SITTING_UPSTAIRS", 369:"SITTING_DOWNSTAIRS", 370:"Lying_Side", 371:"Lying_Front", 372:"Lying_Belly", 373:"WALKING_UPSTAIRS", 374:"WALKING_DOWNSTAIRS", 375:"SITTING_UPSTAIRS", 376:"SITTING_DOWNSTAIRS", 377:"Lying_Side", 378:"Lying_Front", 379:"Lying_Belly", 380:"STANDING_UPSTAIRS", 381:"STANDING_DOWNSTAIRS", 382:"SITTING_UPSTAIRS", 383:"SITTING_DOWNSTAIRS", 384:"Lying_Side", 385:"Lying_Front", 386:"Lying_Belly", 387:"WALKING_UPSTAIRS", 388:"WALKING_DOWNSTAIRS", 389:"SITTING_UPSTAIRS", 390:"SITTING_DOWNSTAIRS", 391:"Lying_Side", 392:"Lying_Front", 393:"Lying_Belly", 394:"STANDING_UPSTAIRS", 395:"STANDING_DOWNSTAIRS", 396:"SITTING_UPSTAIRS", 397:"SITTING_DOWNSTAIRS", 398:"Lying_Side", 399:"Lying_Front", 400:"Lying_Belly", 401:"WALKING_UPSTAIRS", 402:"WALKING_DOWNSTAIRS", 403:"SITTING_UPSTAIRS", 404:"SITTING_DOWNSTAIRS", 405:"Lying_Side", 406:"Lying_Front", 407:"Lying_Belly", 408:"STANDING_UPSTAIRS", 409:"STANDING_DOWNSTAIRS", 410:"SITTING_UPSTAIRS", 411:"SITTING_DOWNSTAIRS", 412:"Lying_Side", 413:"Lying_Front", 414:"Lying_Belly", 415:"WALKING_UPSTAIRS", 416:"WALKING_DOWNSTAIRS", 417:"SITTING_UPSTAIRS", 418:"SITTING_DOWNSTAIRS", 419:"Lying_Side", 420:"Lying_Front", 421:"Lying_Belly", 422:"STANDING_UPSTAIRS", 423:"STANDING_DOWNSTAIRS", 424:"SITTING_UPSTAIRS", 425:"SITTING_DOWNSTAIRS", 426:"Lying_Side", 427:"Lying_Front", 428:"Lying_Belly", 429:"WALKING_UPSTAIRS", 430:"WALKING_DOWNSTAIRS", 431:"SITTING_UPSTAIRS", 432:"SITTING_DOWNSTAIRS", 433:"Lying_Side", 434:"Lying_Front", 435:"Lying_Belly", 436:"STANDING_UPSTAIRS", 437:"STANDING_DOWNSTAIRS", 438:"SITTING_UPSTAIRS", 439:"SITTING_DOWNSTAIRS", 440:"Lying_Side", 441:"Lying_Front", 442:"Lying_Belly", 443:"WALKING_UPSTAIRS", 444:"WALKING_DOWNSTAIRS", 445:"SITTING_UPSTAIRS", 446:"SITTING_DOWNSTAIRS", 447:"Lying_Side", 448:"Lying_Front", 449:"Lying_Belly", 450:"STANDING_UPSTAIRS", 451:"STANDING_DOWNSTAIRS", 452:"SITTING_UPSTAIRS", 453:"SITTING_DOWNSTAIRS", 454:"Lying_Side", 455:"Lying_Front", 456:"Lying_Belly", 457:"WALKING_UPSTAIRS", 458:"WALKING_DOWNSTAIRS", 459:"SITTING_UPSTAIRS", 460:"SITTING_DOWNSTAIRS", 461:"Lying_Side", 462:"Lying_Front", 463:"Lying_Belly", 464:"STANDING_UPSTAIRS", 465:"STANDING_DOWNSTAIRS", 466:"SITTING_UPSTAIRS", 467:"SITTING_DOWNSTAIRS", 468:"Lying_Side", 469:"Lying_Front", 470:"Lying_Belly", 471:"WALKING_UPSTAIRS", 472:"WALKING_DOWNSTAIRS", 473:"SITTING_UPSTAIRS", 474:"SITTING_DOWNSTAIRS", 475:"Lying_Side", 476:"Lying_Front", 477:"Lying_Belly", 478:"STANDING_UPSTAIRS", 479:"STANDING_DOWNSTAIRS", 480:"SITTING_UPSTAIRS", 481:"SITTING_DOWNSTAIRS", 482:"Lying_Side", 483:"Lying_Front", 484:"Lying_Belly", 485:"WALKING_UPSTAIRS", 486:"WALKING_DOWNSTAIRS", 487:"SITTING_UPSTAIRS", 488:"SITTING_DOWNSTAIRS", 489:"Lying_Side", 490:"Lying_Front", 491:"Lying_Belly", 492:"STANDING_UPSTAIRS", 493:"STANDING_DOWNSTAIRS", 494:"SITTING_UPSTAIRS", 495:"SITTING_DOWNSTAIRS", 496:"Lying_Side", 497:"Lying_Front", 498:"Lying_Belly", 499:"WALKING_UPSTAIRS", 500:"WALKING_DOWNSTAIRS", 501:"SITTING_UPSTAIRS", 502:"SITTING_DOWNSTAIRS", 503:"Lying_Side", 504:"Lying_Front", 505:"Lying_Belly", 506:"STANDING_UPSTAIRS", 507:"STANDING_DOWNSTAIRS", 508:"SITTING_UPSTAIRS", 509:"SITTING_DOWNSTAIRS", 510:"Lying_Side", 511:"Lying_Front", 512:"Lying_Belly", 513:"WALKING_UPSTAIRS", 514:"WALKING_DOWNSTAIRS", 515:"SITTING_UPSTAIRS", 516:"SITTING_DOWNSTAIRS", 517:"Lying_Side", 518:"Lying_Front", 519:"Lying_Belly", 520:"STANDING_UPSTAIRS", 521:"STANDING_DOWNSTAIRS", 522:"SITTING_UPSTAIRS", 523:"SITTING_DOWNSTAIRS", 524:"Lying_Side", 525:"Lying_Front", 526:"Lying_Belly", 527:"WALKING_UPSTAIRS", 528:"WALKING_DOWNSTAIRS", 529:"SITTING_UPSTAIRS", 530:"SITTING_DOWNSTAIRS", 531:"Lying_Side", 532:"Lying_Front", 533:"Lying_Belly", 534:"STANDING_UPSTAIRS", 535:"STANDING_DOWNSTAIRS", 536:"SITTING_UPSTAIRS", 537:"SITTING_DOWNSTAIRS", 538:"Lying_Side", 539:"Lying_Front", 540:"Lying_Belly", 541:"WALKING_UPSTAIRS", 542:"WALKING_DOWNSTAIRS", 543:"SITTING_UPSTAIRS", 544:"SITTING_DOWNSTAIRS", 545:"Lying_Side", 546:"Lying_Front", 547:"Lying_Belly", 548:"STANDING_UPSTAIRS", 549:"STANDING_DOWNSTAIRS", 550:"SITTING_UPSTAIRS", 551:"SITTING_DOWNSTAIRS", 552:"Lying_Side", 553:"Lying_Front", 554:"Lying_Belly", 555:"WALKING_UPSTAIRS", 556:"WALKING_DOWNSTAIRS", 557:"SITTING_UPSTAIRS", 558:"SITTING_DOWNSTAIRS", 559:"Lying_Side", 560:"Lying_Front", 561:"Lying_Belly", 562:"STANDING_UPSTAIRS", 563:"STANDING_DOWNSTAIRS", 564:"SITTING_UPSTAIRS", 565:"SITTING_DOWNSTAIRS", 566:"Lying_Side", 567:"Lying_Front", 568:"Lying_Belly", 569:"WALKING_UPSTAIRS", 570:"WALKING_DOWNSTAIRS", 571:"SITTING_UPSTAIRS", 572:"SITTING_DOWNSTAIRS", 573:"Lying_Side", 574:"Lying_Front", 575:"Lying_Belly", 576:"STANDING_UPSTAIRS", 577:"STANDING_DOWNSTAIRS", 578:"SITTING_UPSTAIRS", 579:"SITTING_DOWNSTAIRS", 580:"Lying_Side", 581:"Lying_Front", 582:"Lying_Belly", 583:"WALKING_UPSTAIRS", 584:"WALKING_DOWNSTAIRS", 585:"SITTING_UPSTAIRS", 586:"SITTING_DOWNSTAIRS", 587:"Lying_Side", 588:"Lying_Front", 589:"Lying_Belly", 590:"STANDING_UPSTAIRS", 591:"STANDING_DOWNSTAIRS", 592:"SITTING_UPSTAIRS", 593:"SITTING_DOWNSTAIRS", 594:"Lying_Side", 595:"Lying_Front", 596:"Lying_Belly", 597:"WALKING_UPSTAIRS", 598:"WALKING_DOWNSTAIRS", 599:"SITTING_UPSTAIRS", 600:"SITTING_DOWNSTAIRS", 601:"Lying_Side", 602:"Lying_Front", 603:"Lying_Belly", 604:"STANDING_UPSTAIRS", 605:"STANDING_DOWNSTAIRS", 606:"SITTING_UPSTAIRS", 607:"SITTING_DOWNSTAIRS", 608:"Lying_Side", 609:"Lying_Front", 610:"Lying_Belly", 611:"WALKING_UPSTAIRS", 612:"WALKING_DOWNSTAIRS", 613:"SITTING_UPSTAIRS", 614:"SITTING_DOWNSTAIRS", 615:"Lying_Side", 616:"Lying_Front", 617:"Lying_Belly", 618:"STANDING_UPSTAIRS", 619:"STANDING_DOWNSTAIRS", 620:"SITTING_UPSTAIRS", 621:"SITTING_DOWNSTAIRS", 622:"Lying_Side", 623:"Lying_Front", 624:"Lying_Belly", 625:"WALKING_UPSTAIRS", 626:"WALKING_DOWNSTAIRS", 627:"SITTING_UPSTAIRS", 628:"SITTING_DOWNSTAIRS", 629:"Lying_Side", 630:"Lying_Front", 631:"Lying_Belly", 632:"STANDING_UPSTAIRS", 633:"STANDING_DOWNSTAIRS", 634:"SITTING_UPSTAIRS", 635:"SITTING_DOWNSTAIRS", 636:"Lying_Side", 637:"Lying_Front", 638:"Lying_Belly", 639:"WALKING_UPSTAIRS", 640:"WALKING_DOWNSTAIRS", 641:"SITTING_UPSTAIRS", 642:"SITTING_DOWNSTAIRS", 643:"Lying_Side", 644:"Lying_Front", 645:"Lying_Belly", 646:"STANDING_UPSTAIRS", 647:"STANDING_DOWNSTAIRS", 648:"SITTING_UPSTAIRS", 649:"SITTING_DOWNSTAIRS", 650:"Lying_Side", 651:"Lying_Front", 652:"Lying_Belly", 653:"WALKING_UPSTAIRS", 654:"WALKING_DOWNSTAIRS", 655:"SITTING_UPSTAIRS", 656:"SITTING_DOWNSTAIRS", 657:"Lying_Side", 658:"Lying_Front", 659:"Lying_Belly", 660:"STANDING_UPSTAIRS", 661:"STANDING_DOWNSTAIRS", 662:"SITTING_UPSTAIRS", 663:"SITTING_DOWNSTAIRS", 664:"Lying_Side", 665:"Lying_Front", 666:"Lying_Belly", 667:"WALKING_UPSTAIRS", 668:"WALKING_DOWNSTAIRS", 669:"SITTING_UPSTAIRS", 670:"SITTING_DOWNSTAIRS", 671:"Lying_Side", 672:"Lying_Front", 673:"Lying_Belly", 674:"STANDING_UPSTAIRS", 675:"STANDING_DOWNSTAIRS", 676:"SITTING_UPSTAIRS", 677:"SITTING_DOWNSTAIRS", 678:"Lying_Side", 679:"Lying_Front", 680:"Lying_Belly", 681:"WALKING_UPSTAIRS", 682:"WALKING_DOWNSTAIRS", 683:"SITTING_UPSTAIRS", 684:"SITTING_DOWNSTAIRS", 685:"Lying_Side", 686:"Lying_Front", 687:"Lying_Belly", 688:"STANDING_UPSTAIRS", 689:"STANDING_DOWNSTAIRS", 690:"SITTING_UPSTAIRS", 691:"SITTING_DOWNSTAIRS", 692:"Lying_Side", 693:"Lying_Front", 694:"Lying_Belly", 695:"WALKING_UPSTAIRS", 696:"WALKING_DOWNSTAIRS", 697:"SITTING_UPSTAIRS", 698:"SITTING_DOWNSTAIRS", 699:"Lying_Side", 700:"Lying_Front", 701:"Lying_Belly", 702:"STANDING_UPSTAIRS", 703:"STANDING_DOWNSTAIRS", 704:"SITTING_UPSTAIRS", 705:"SITTING_DOWNSTAIRS", 706:"Lying_Side", 707:"Lying_Front", 708:"Lying_Belly", 709:"WALKING_UPSTAIRS", 710:"WALKING_DOWNSTAIRS", 711:"SITTING_UPSTAIRS", 712:"SITTING_DOWNSTAIRS", 713:"Lying_Side", 714:"Lying_Front", 715:"Lying_Belly", 716:"STANDING_UPSTAIRS", 717:"STANDING_DOWNSTAIRS", 718:"SITTING_UPSTAIRS", 719:"SITTING_DOWNSTAIRS", 720:"Lying_Side", 721:"Lying_Front", 722:"Lying_Belly", 723:"WALKING_UPSTAIRS", 724:"WALKING_DOWNSTAIRS", 725:"SITTING_UPSTAIRS", 726:"SITTING_DOWNSTAIRS", 727:"Lying_Side", 728:"Lying_Front", 729:"Lying_Belly", 730:"STANDING_UPSTAIRS", 731:"STANDING_DOWNSTAIRS", 732:"SITTING_UPSTAIRS", 733:"SITTING_DOWNSTAIRS", 734:"Lying_Side", 735:"Lying_Front", 736:"Lying_Belly", 737:"WALKING_UPSTAIRS", 738:"WALKING_DOWNSTAIRS", 739:"SITTING_UPSTAIRS", 740:"SITTING_DOWNSTAIRS", 741:"Lying_Side", 742:"Lying_Front", 743:"Lying_Belly", 744:"STANDING_UPSTAIRS", 745:"STANDING_DOWNSTAIRS", 746:"SITTING_UPSTAIRS", 747:"SITTING_DOWNSTAIRS", 748:"Lying_Side", 749:"Lying_Front", 750:"Lying_Belly", 751:"WALKING_UPSTAIRS", 752:"WALKING_DOWNSTAIRS", 753:"SITTING_UPSTAIRS", 754:"SITTING_DOWNSTAIRS", 755:"Lying_Side", 756:"Lying_Front", 757:"Lying_Belly", 758:"STANDING_UPSTAIRS", 759:"STANDING_DOWNSTAIRS", 760:"SITTING_UPSTAIRS", 761:"SITTING_DOWNSTAIRS", 762:"Lying_Side", 763:"Lying_Front", 764:"Lying_Belly", 765:"WALKING_UPSTAIRS", 766:"WALKING_DOWNSTAIRS", 767:"SITTING_UPSTAIRS", 768:"SITTING_DOWNSTAIRS", 769:"Lying_Side", 770:"Lying_Front", 771:"Lying_Belly", 772:"STANDING_UPSTAIRS", 773:"STANDING_DOWNSTAIRS", 774:"SITTING_UPSTAIRS", 775:"SITTING_DOWNSTAIRS", 776:"Lying_Side", 777:"Lying_Front", 778:"Lying_Belly", 779:"WALKING_UPSTAIRS", 780:"WALKING_DOWNSTAIRS", 781:"SITTING_UPSTAIRS", 782:"SITTING_DOWNSTAIRS", 783:"Lying_Side", 784:"Lying_Front", 785:"Lying_Belly", 786:"STANDING_UPSTAIRS", 787:"STANDING_DOWNSTAIRS", 788:"SITTING_UPSTAIRS", 789:"SITTING_DOWNSTAIRS", 790:"Lying_Side", 791:"Lying_Front", 792:"Lying_Belly", 793:"WALKING_UPSTAIRS", 794:"WALKING_DOWNSTAIRS", 795:"SITTING_UPSTAIRS", 796:"SITTING_DOWNSTAIRS", 797:"Lying_Side", 798:"Lying_Front", 799:"Lying_Belly", 800:"STANDING_UPSTAIRS", 801:"STANDING_DOWNSTAIRS", 802:"SITTING_UPSTAIRS", 803:"SITTING_DOWNSTAIRS", 804:"Lying_Side", 805:"Lying_Front", 806:"Lying_Belly", 807:"WALKING_UPSTAIRS", 808:"WALKING_DOWNSTAIRS", 809:"SITTING_UPSTAIRS", 810:"SITTING_DOWNSTAIRS", 811:"Lying_Side", 812:"Lying_Front", 813:"Lying_Belly", 814:"STANDING_UPSTAIRS", 815:"STANDING_DOWNSTAIRS", 816:"SITTING_UPSTAIRS", 817:"SITTING_DOWNSTAIRS", 818:"Lying_Side", 819:"Lying_Front", 820:"Lying_Belly", 821:"WALKING_UPSTAIRS", 822:"WALKING_DOWNSTAIRS", 823:"SITTING_UPSTAIRS", 824:"SITTING_DOWNSTAIRS", 825:"Lying_Side", 826:"Lying_Front", 827:"Lying_Belly", 828:"STANDING_UPSTAIRS", 829:"STANDING_DOWNSTAIRS", 830:"SITTING_UPSTAIRS", 831:"SITTING_DOWNSTAIRS", 832:"Lying_Side", 833:"Lying_Front", 834:"Lying_Belly", 835:"WALKING_UPSTAIRS", 836:"WALKING_DOWNSTAIRS", 837:"SITTING_UPSTAIRS", 838:"SITTING_DOWNSTAIRS", 839:"Lying_Side", 840:"Lying_Front", 841:"Lying_Belly", 842:"STANDING_UPSTAIRS", 843:"STANDING_DOWNSTAIRS", 844:"SITTING_UPSTAIRS", 845:"SITTING_DOWNSTAIRS", 846:"Lying_Side", 847:"Lying_Front", 848:"Lying_Belly", 849:"WALKING_UPSTAIRS", 850:"WALKING_DOWNSTAIRS", 851:"SITTING_UPSTAIRS", 852:"SITTING_DOWNSTAIRS", 853:"Lying_Side", 854:"Lying_Front", 855:"Lying_Belly", 856:"STANDING_UPSTAIRS", 857:"STANDING_DOWNSTAIRS", 858:"SITTING_UPSTAIRS", 859:"SITTING_DOWNSTAIRS", 860:"Lying_Side", 861:"Lying_Front", 862:"Lying_Belly", 863:"WALKING_UPSTAIRS", 864:"WALKING_DOWNSTAIRS", 865:"SITTING_UPSTAIRS", 866:"SITTING_DOWNSTAIRS", 867:"Lying_Side", 868:"Lying_Front", 869:"Lying_Belly", 870:"STANDING_UPSTAIRS", 871:"STANDING_DOWNSTAIRS", 872:"SITTING_UPSTAIRS", 873:"SITTING_DOWNSTAIRS", 874:"Lying_Side", 875:"Lying_Front", 876:"Lying_Belly", 877:"WALKING_UPSTAIRS", 878:"WALKING_DOWNSTAIRS", 879:"SITTING_UPSTAIRS", 880:"SITTING_DOWNSTAIRS", 881:"Lying_Side", 882:"Lying_Front", 883:"Lying_Belly", 884:"STANDING_UPSTAIRS", 885:"STANDING_DOWNSTAIRS", 886:"SITTING_UPSTAIRS", 887:"SITTING_DOWNSTAIRS", 888:"Lying_Side", 889:"Lying_Front", 890:"Lying_Belly", 891:"WALKING_UPSTAIRS", 892:"WALKING_DOWNSTAIRS", 893:"SITTING_UPSTAIRS", 894:"SITTING_DOWNSTAIRS", 895:"Lying_Side", 896:"Lying_Front", 897:"Lying_Belly", 898:"STANDING_UPSTAIRS", 899:"STANDING_DOWNSTAIRS", 900:"SITTING_UPSTAIRS", 901:"SITTING_DOWNSTAIRS", 902:"Lying_Side", 903:"Lying_Front", 904:"Lying_Belly", 905:"WALKING_UPSTAIRS", 906:"WALKING_DOWNSTAIRS", 907:"SITTING_UPSTAIRS", 908:"SITTING_DOWNSTAIRS", 909:"Lying_Side", 910:"Lying_Front", 911:"Lying_Belly", 912:"STANDING_UPSTAIRS", 913:"STANDING_DOWNSTAIRS", 914:"SITTING_UPSTAIRS", 915:"SITTING_DOWNSTAIRS", 916:"Lying_Side", 917:"Lying_Front", 918:"Lying_Belly", 919:"WALKING_UPSTAIRS", 920:"WALKING_DOWNSTAIRS", 921:"SITTING_UPSTAIRS", 922:"SITTING_DOWNSTAIRS", 923:"Lying_Side", 924:"Lying_Front", 925:"Lying_Belly", 926:"STANDING_UPSTAIRS", 927:"STANDING_DOWNSTAIRS", 928:"SITTING_UPSTAIRS", 929:"SITTING_DOWNSTAIRS", 930:"Lying
```

```
In [4]: 1 #Check if data has any null values.  
2 train_data.isnull().sum().sum()
```

Out[4]: 0

```
In [ ]: 1 #Print any features with null values. Not used now. Same as above.  
2 nulls = train_data.isnull().sum().to_frame()  
3 for index, row in nulls.iterrows():  
4     if row[0]>0:print(index, row[0])
```

```
In [5]: 1 #Study different acitivity sample data differences. Mean and variance selected.  
2 train_data.groupby('Activity').agg(['mean', 'var'])
```

Out[5]:

	1 tBodyAcc-mean()-X	2 tBodyAcc-mean()-Y	3 tBodyAcc-mean()-Z	4 tBodyAcc-std()-X	5 tBodyAcc-std()-Y	... 551				
Activity	mean	var	mean	var	mean	var	mean	var	... me	
1	0.276259	0.002536	-0.017767	0.000436	-0.108892	0.001052	-0.312640	0.020869	-0.020279	0.031568
2	0.261925	0.006090	-0.026648	0.001372	-0.120423	0.003625	-0.221069	0.023843	-0.000344	0.040554
3	0.288170	0.009044	-0.016368	0.000732	-0.105860	0.002566	0.139856	0.049337	0.079190	0.059430
4	0.273450	0.001764	-0.012142	0.001050	-0.106576	0.002054	-0.983440	0.001042	-0.936213	0.016919
5	0.279298	0.000404	-0.016124	0.000319	-0.107327	0.001273	-0.985346	0.000523	-0.936012	0.006714
6	0.269187	0.010310	-0.018345	0.005405	-0.107178	0.008057	-0.959476	0.006196	-0.937598	0.021585

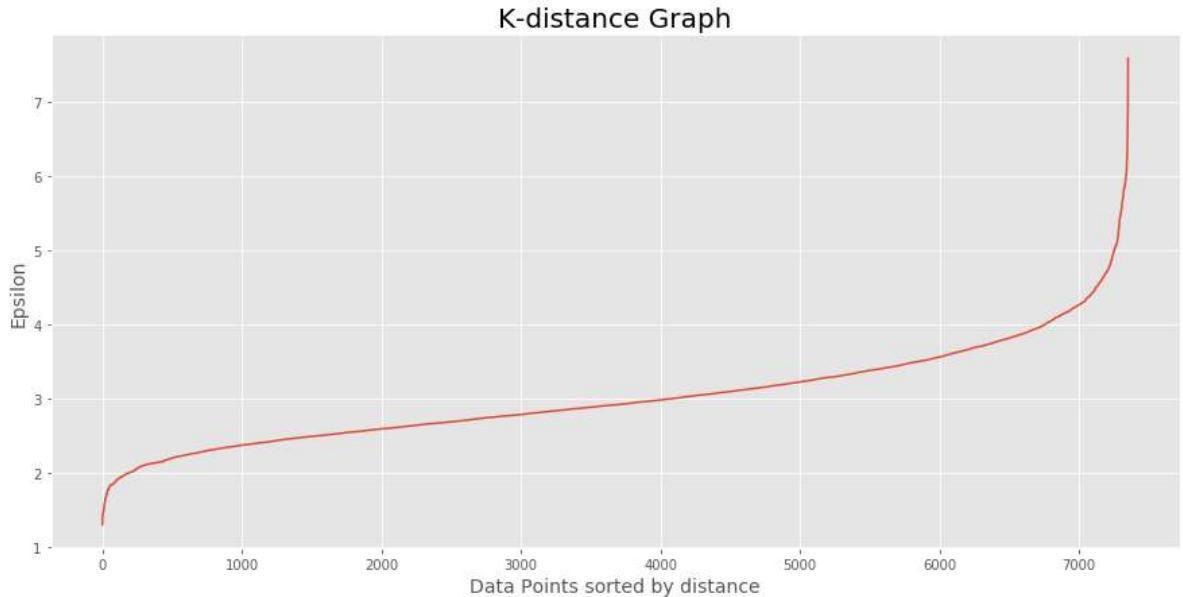
6 rows × 1122 columns

```
In [7]: 1 train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 7352 entries, 0 to 7351  
Columns: 562 entries, 1 tBodyAcc-mean()-X to Activity  
dtypes: float64(561), int64(1)  
memory usage: 31.5 MB
```

Use the input data without dimensionality reduction.

```
In [8]: #Plot nearest neighbor graph for distance parameter setting.
1 neigh = NearestNeighbors(n_neighbors=10)
2 nbrs = neigh.fit(X)
3 distances, indices = nbrs.kneighbors(X)
4 # Plotting K-distance Graph
5 distances = np.sort(distances, axis=0)
6 distances = distances[:,1]
7 plt.plot(distances)
8 plt.title('K-distance Graph', fontsize=20)
9 plt.xlabel('Data Points sorted by distance', fontsize=14)
10 plt.ylabel('Epsilon', fontsize=14)
11 plt.show()
```



```
In [10]: #Set range of values using above graph. This to fond optimal DBSCAN parameters
1 eps_values = np.arange(3.6,6,0.5) # with normalization and PCA
2 min_samples = np.arange(4,16,2) # with normalization and PCA 20,40,2
3 DBSCAN_params = list(product(eps_values, min_samples))
4 train_data.drop(columns=['Activity'])
```

Out[10]:

	1 tBodyAcc-mean()-X	2 tBodyAcc-mean()-Y	3 tBodyAcc-mean()-Z	4 tBodyAcc-std()-X	5 tBodyAcc-std()-Y	6 tBodyAcc-std()-Z	7 tBodyAcc-mad()-X	8 tBodyAcc-mad()-Y	9 tBodyAcc-mad()-Z	10 tBodyAcc-max()
0	0.289	-0.02030	-0.133	-0.995	-0.98300	-0.9140	-0.995	-0.9830	-0.9240	-0.5
1	0.278	-0.01640	-0.124	-0.998	-0.97500	-0.9600	-0.999	-0.9750	-0.9580	-0.5
2	0.280	-0.01950	-0.113	-0.995	-0.96700	-0.9790	-0.997	-0.9640	-0.9770	-0.5
3	0.279	-0.02620	-0.123	-0.996	-0.98300	-0.9910	-0.997	-0.9830	-0.9890	-0.5
4	0.277	-0.01660	-0.115	-0.998	-0.98100	-0.9900	-0.998	-0.9800	-0.9900	-0.5
...
7347	0.300	-0.05720	-0.181	-0.195	0.03990	0.0771	-0.282	0.0436	0.0604	0.2
7348	0.274	-0.00775	-0.147	-0.235	0.00482	0.0593	-0.323	-0.0295	0.0806	0.1
7349	0.273	-0.01700	-0.045	-0.218	-0.10400	0.2750	-0.305	-0.0989	0.3330	0.0
7350	0.290	-0.01880	-0.158	-0.219	-0.11100	0.2690	-0.310	-0.0682	0.3190	0.1
7351	0.352	-0.01240	-0.204	-0.269	-0.08720	0.1770	-0.377	-0.0387	0.2290	0.2

7352 rows × 561 columns

```

In [11]: # THIS CAN TAKE ANYTHING FROM 30 MINUTES TO OVER ONE HOUR TO RUN!!! NOT NEEDED AS OPTIMAL PARA
1 #Run DBSCAN fit with multiple parameter sets to find optimal parameters for best most accurate
2 #Run timer to check computation time.
3
4 no_of_clusters = []
5 sil_score = []
6 print('PCU utilization:', psutil.cpu_percent(), '%')
7 print('Clusters created at each round. Looking for 6 with high silhouette score. Will take about')
8 start = time.time()
9
10 for p in DBSCAN_params:
11     DBS_clustering = DBSCAN(eps=p[0], min_samples=p[1]).fit(train_data)
12     no_of_clusters.append(len(np.unique(DBS_clustering.labels_)))
13     #print(DBS_clustering.labels_)
14     #print(no_of_clusters)
15     #print(p[0],p[1])
16     #print('PCU utilization%:', psutil.cpu_percent())
17     if len(np.unique(DBS_clustering.labels_)) > 1:
18         sil_score.append(silhouette_score(train_data, DBS_clustering.labels_))
19         good_label = DBS_clustering.labels_
20     else:
21         sil_score.append(silhouette_score(train_data, good_label))
22
23 end = time.time()
24 print('PCU utilization:', psutil.cpu_percent(), '%')
25 print('Total time: %.2f' % (end - start), 's')

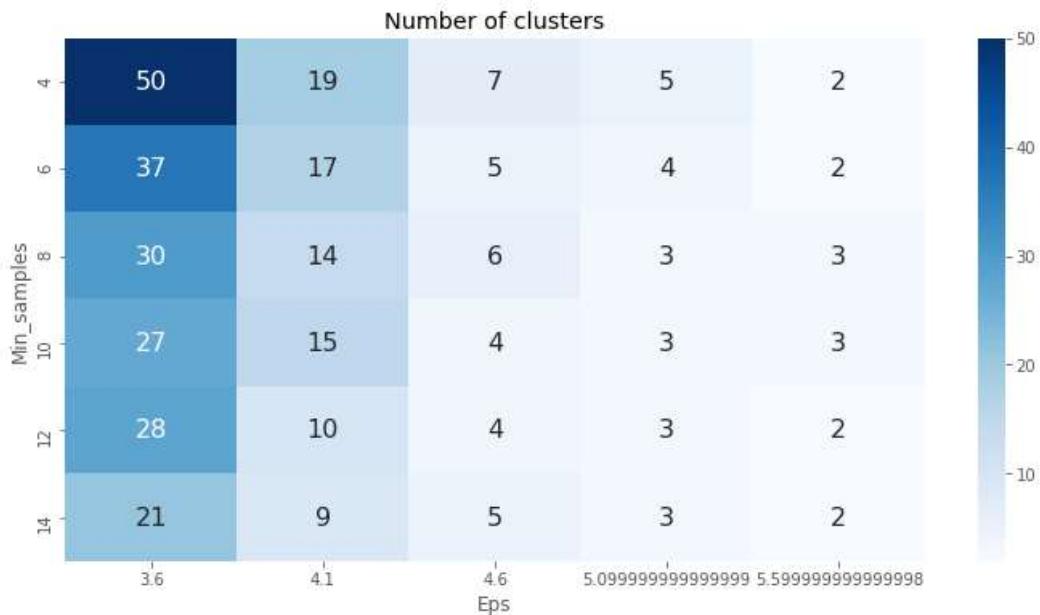
```

PCU utilization: 23.3 %
 Clusters created at each round. Looking for 6 with high silhouette score. Will take about 1800 to 3600s to run.
 PCU utilization: 56.3 %
 Total time: 2970.56 s

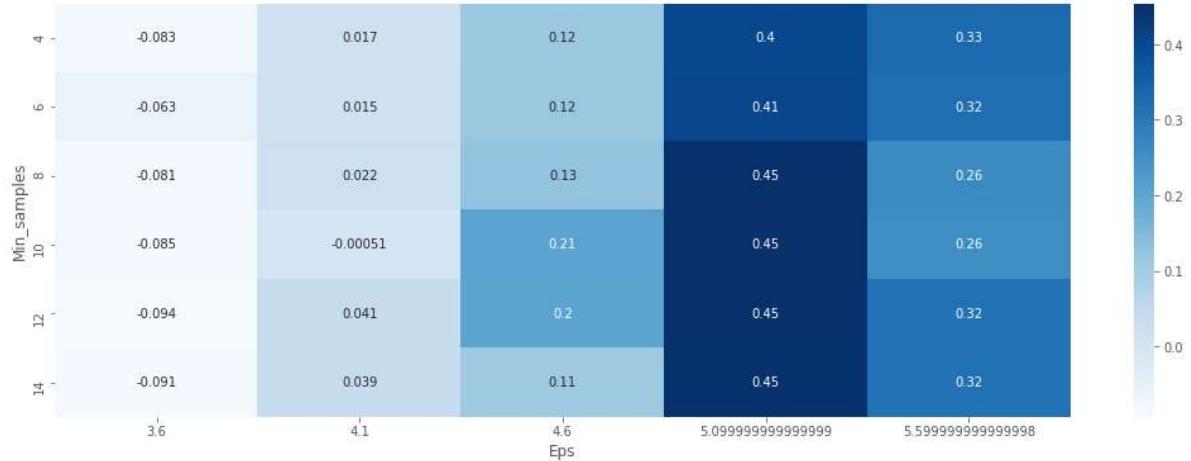
```

In [12]: #Show clustering matrix with number of clusters, eps and min_samples (x,y).
1 tmp = pd.DataFrame.from_records(DBSCAN_params, columns=['Eps', 'Min_samples'])
2 tmp['No_of_clusters'] = no_of_clusters
3 pivot_1 = pd.pivot_table(tmp, values='No_of_clusters', index='Min_samples', columns='Eps')
4 fig, ax = plt.subplots(figsize=(12,6))
5 sns.heatmap(pivot_1, annot=True, annot_kws={"size": 16}, cmap="Blues", ax=ax)
6 ax.set_title('Number of clusters')
7 plt.show()

```



```
In [13]: #Show clustering matrix with silhouette score, eps and min_samples (x,y).
#Silhouette score indicates how well clusters are separates. Range is -1..1. 1 is the best score
tmp = pd.DataFrame.from_records(DBSCAN_params, columns=['Eps', 'Min_samples'])
tmp['Sil_score'] = sil_score
pivot_1 = pd.pivot_table(tmp, values='Sil_score', index='Min_samples', columns='Eps')
fig, ax = plt.subplots(figsize=(18,6))
sns.heatmap(pivot_1, annot=True, annot_kws={"size": 10}, cmap="Blues", ax=ax)
plt.show()
```



```
In [14]: #Final clustering with most optimal eps and min_samples parameters.
print('PCU utilization:', psutil.cpu_percent(), '%')
start = time.time()
DBS_clustering = DBSCAN(eps=4.5, min_samples=10).fit(X)
core_samples_mask = np.zeros_like(DBS_clustering.labels_, dtype=bool)
core_samples_mask[DBS_clustering.core_sample_indices_] = True
labels = DBS_clustering.labels_
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noisy_ = list(labels).count(-1)
DBSCAN_clustered = pd.DataFrame(X.copy())
DBSCAN_clustered.loc[:, 'Cluster'] = DBS_clustering.labels_ # append Labels to points
end = time.time()
print('PCU utilization:', psutil.cpu_percent(), '%')
print('Total time: %0.2f' % (end-start), 's')
```

PCU utilization: 34.9 %
 PCU utilization: 74.1 %
 Total time: 160.94 s

```
In [15]: #Check number of clusters and amount of samples in each. DBSCAN only able to create two clusters
DBSCAN_clust_sizes = DBSCAN_clustered.groupby('Cluster').size().to_frame()
DBSCAN_clust_sizes.columns = ["DBSCAN_size"]
DBSCAN_clust_sizes
```

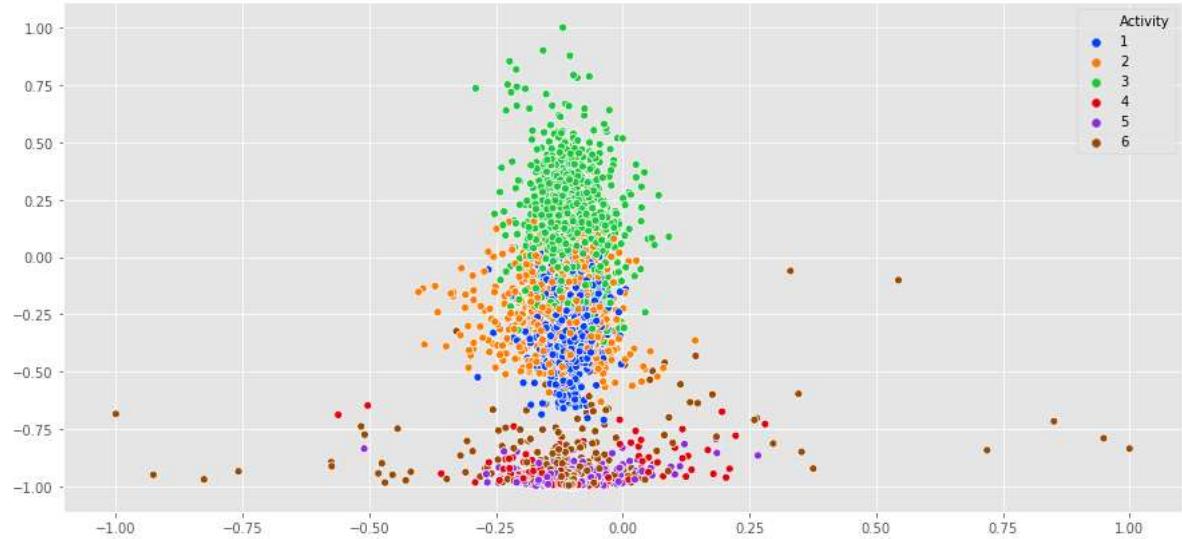
Out[15]:

DBSCAN_size

Cluster	DBSCAN_size
-1	499
0	3850
1	2970
2	33

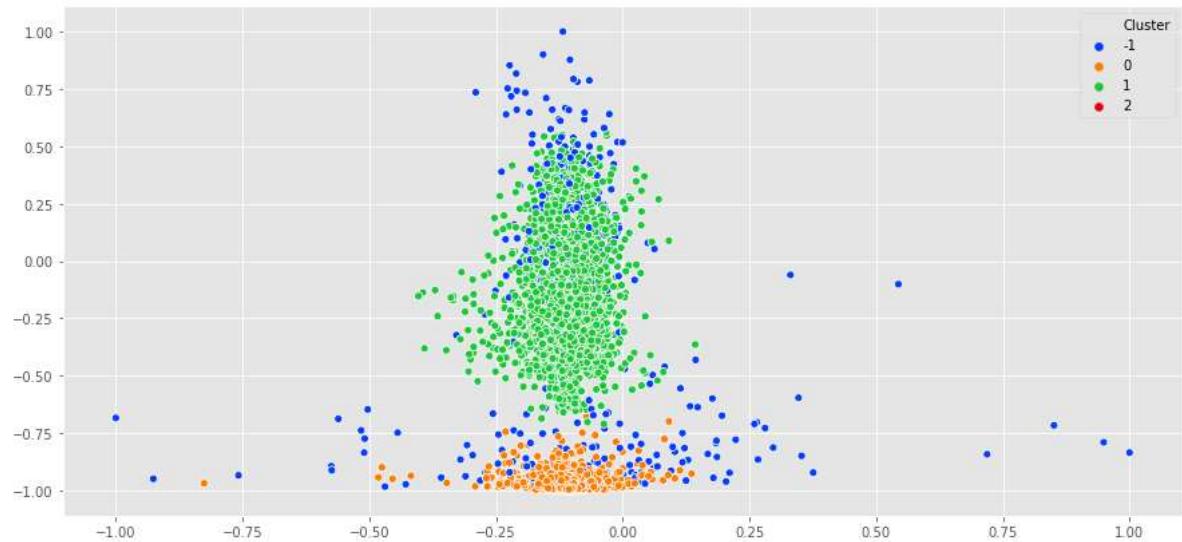
```
In [16]: 1 #Plot the original cluster assignments using given cluster assignments.  
2 sns.scatterplot(X[:,2], X[:,3], hue=train_data["Activity"], palette="bright")
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x231a40281c8>



```
In [17]: 1 #Plot DBSCAN predicted clusters. Label '-1' is outliers. Only two clusters identified.  
2 sns.scatterplot(X[:,2], X[:,3], DBSCAN_clustered['Cluster'], palette="bright")
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x231ab6578c8>



```
In [18]: 1 #Print the DBSCAN KPIs.  
2 label_opt = np.array(np.reshape(label,-1))  
3 label_opt = np.array(np.reshape(label_opt,-1))  
4 label_opt  
5 print('Estimated number of clusters: %d' % n_clusters_)  
6 print('Estimated number of noise points: %d' % n_noise_)  
7 print("Homogeneity: %0.3f" % metrics.homogeneity_score(label_opt, labels))  
8 print("Completeness: %0.3f" % metrics.completeness_score(label_opt, labels))  
9 print("V-measure: %0.3f" % metrics.v_measure_score(label_opt, labels))  
10 print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(label_opt, labels))  
11 print("Adjusted Mutual Information: %0.3f" % metrics.adjusted_mutual_info_score(label_opt, la
```

Estimated number of clusters: 3
Estimated number of noise points: 499
Homogeneity: 0.383
Completeness: 0.749
V-measure: 0.507
Adjusted Rand Index: 0.325
Adjusted Mutual Information: 0.507
Silhouette Coefficient: 0.197

Dimensionality reduction with LDA.

```
In [19]: 1 #Dimensionality reduction. Also called data extraction.  
2 #Transform data to four components. This covers 99% of the variance. See variance ratio below  
3 lda = LDA(n_components=4)  
4 X_train = lda.fit_transform(X, label)  
5 #X_train = lda.fit(X, label).transform(X)
```

C:\Users\marpulli\Anaconda3\lib\site-packages\sklearn\utils\validation.py:760: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
y = column_or_1d(y, warn=True)

```
In [20]: 1 pd.DataFrame(X_train).describe()
```

Out[20]:

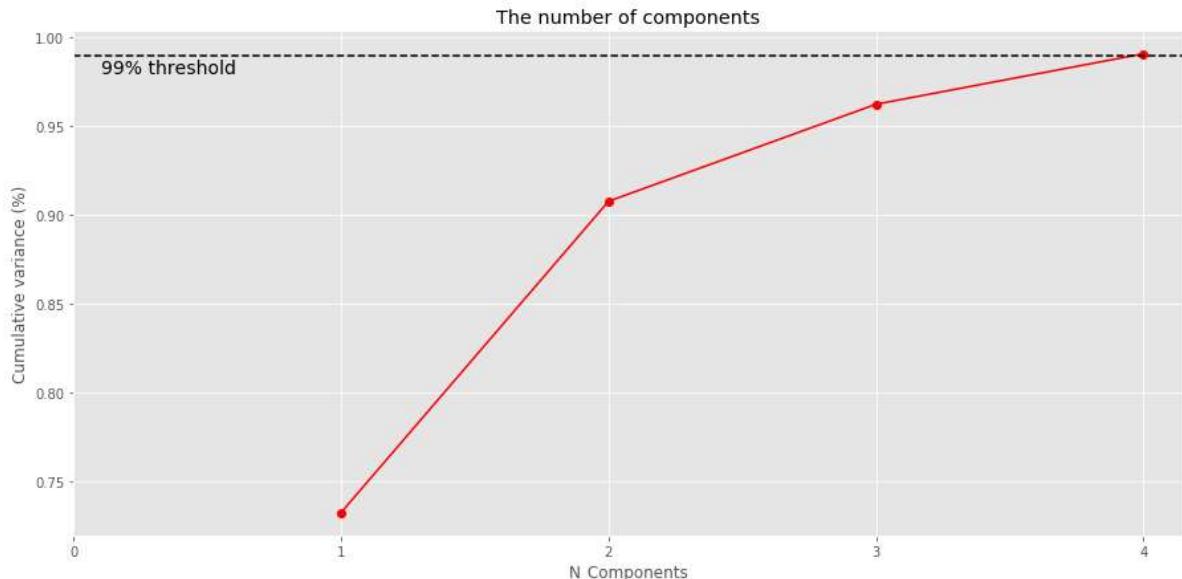
	0	1	2	3
count	7.352000e+03	7.352000e+03	7.352000e+03	7.352000e+03
mean	4.546048e-13	-7.486382e-14	1.874711e-13	-8.228927e-14
std	1.333182e+01	6.575960e+00	3.764320e+00	2.802709e+00
min	-1.924833e+01	-9.313733e+00	-1.110858e+01	-9.866220e+00
25%	-1.436734e+01	-7.036100e+00	-6.568213e-01	-5.025589e-01
50%	9.050437e+00	7.284046e-01	1.048651e-01	2.285788e-01
75%	1.027552e+01	2.084636e+00	7.346217e-01	1.607426e+00
max	2.125605e+01	1.452409e+01	1.123737e+01	7.138333e+00

```
In [21]: 1 #Check how much variance is covered with each component. With four components 99%.  
2 lda.explained_variance_ratio_
```

Out[21]: array([0.73276802, 0.17514607, 0.05460691, 0.02842483])

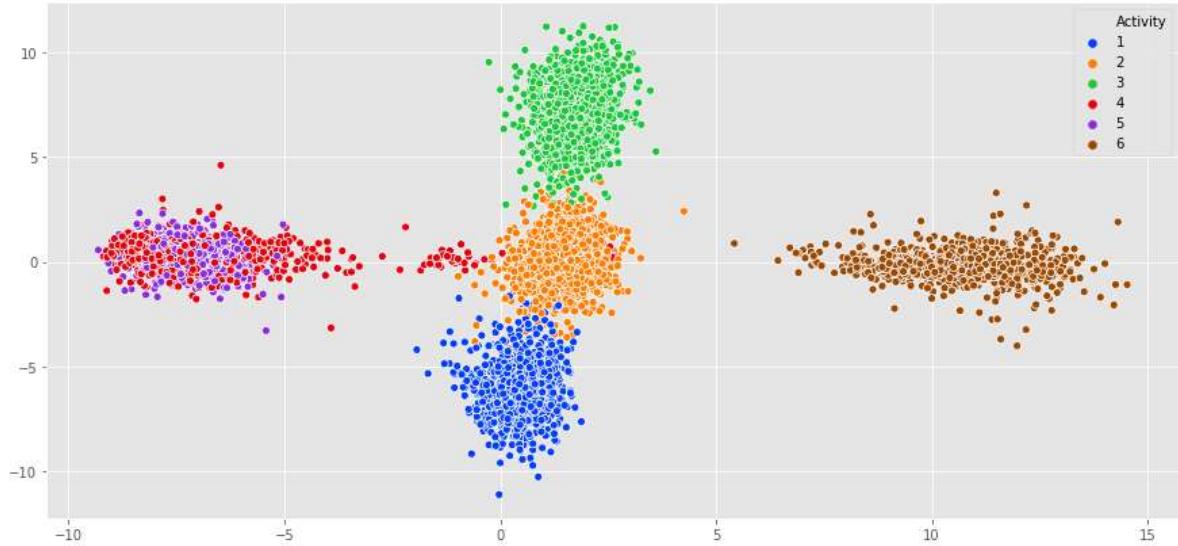
```
In [22]: 1 #Cumulative variance graph  
2 y = np.cumsum(lda.explained_variance_ratio_)  
3 xn = np.arange(1, 5, step=1)  
4 plt.plot(xn, y, marker='o', color='r')  
5 plt.axhline(y=0.99, color='black', linestyle='--')  
6 plt.xticks(np.arange(0, 5, step=1))  
7 plt.xlabel('N_Components')  
8 plt.ylabel('Cumulative variance (%)')  
9 plt.title('The number of components')  
10 plt.text(0.1, 0.98, '99% threshold', color = 'black', fontsize=14)
```

Out[22]: Text(0.1, 0.98, '99% threshold')



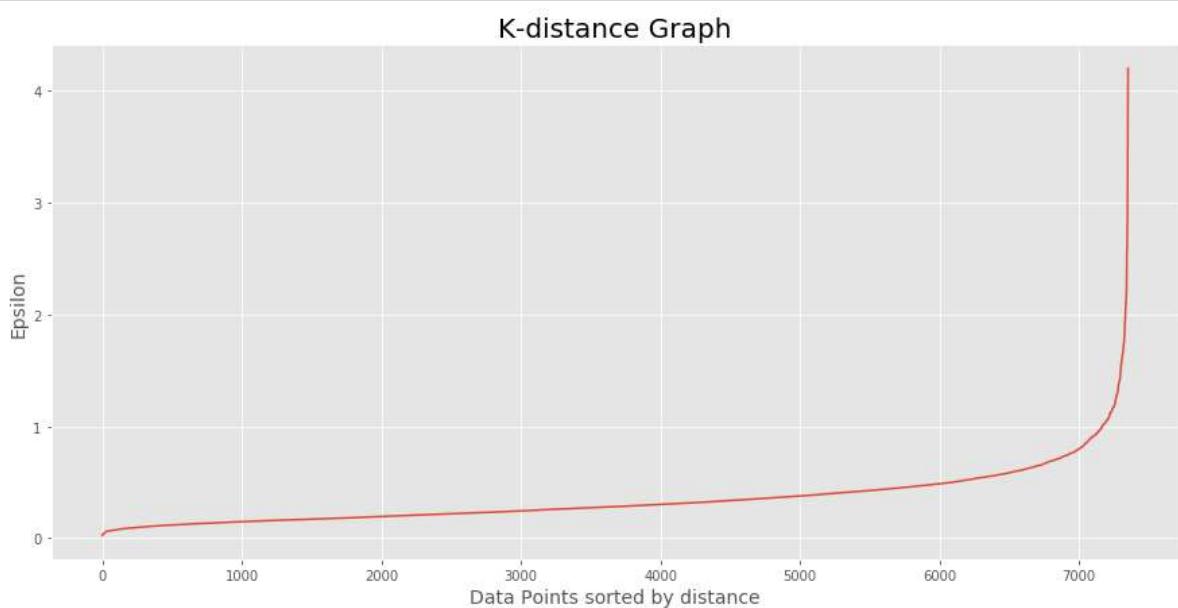
```
In [23]: 1 #Plot the data after dimensionality reduction with LDA data extraction.  
2 sns.scatterplot(X_train[:,1], X_train[:,2], hue=train_data['Activity'], palette='bright')
```

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x231b718fe48>



```
In [24]: 1 #Create nearest neighbor distances.  
2 #X_data = StandardScaler().fit_transform(X_train)  
3 X_data = X_train #No scaling after LDA.  
4 neigh = NearestNeighbors(n_neighbors=4)  
5 nbrs = neigh.fit(X_data)  
6 distances, indices = nbrs.kneighbors(X_data)
```

```
In [25]: 1 # Plotting K-distance graph to find best DBSCAN eps value.  
2 distances = np.sort(distances, axis=0)  
3 distances = distances[:,1]  
4 plt.plot(distances)  
5 plt.title('K-distance Graph', fontsize=20)  
6 plt.xlabel('Data Points sorted by distance', fontsize=14)  
7 plt.ylabel('Epsilon', fontsize=14)  
8 plt.show()
```



```
In [26]: 1 #Set range of values using above graph. This to fond optimal DBSCAN parameters  
2 eps_values = np.arange(0.9,1.9,0.1) # no scaling + LDA these are good  
3 min_samples = np.arange(4,18,2) # no scaling +LDA 4...18 these are good  
4 DBSCAN_params = list(product(eps_values, min_samples))
```

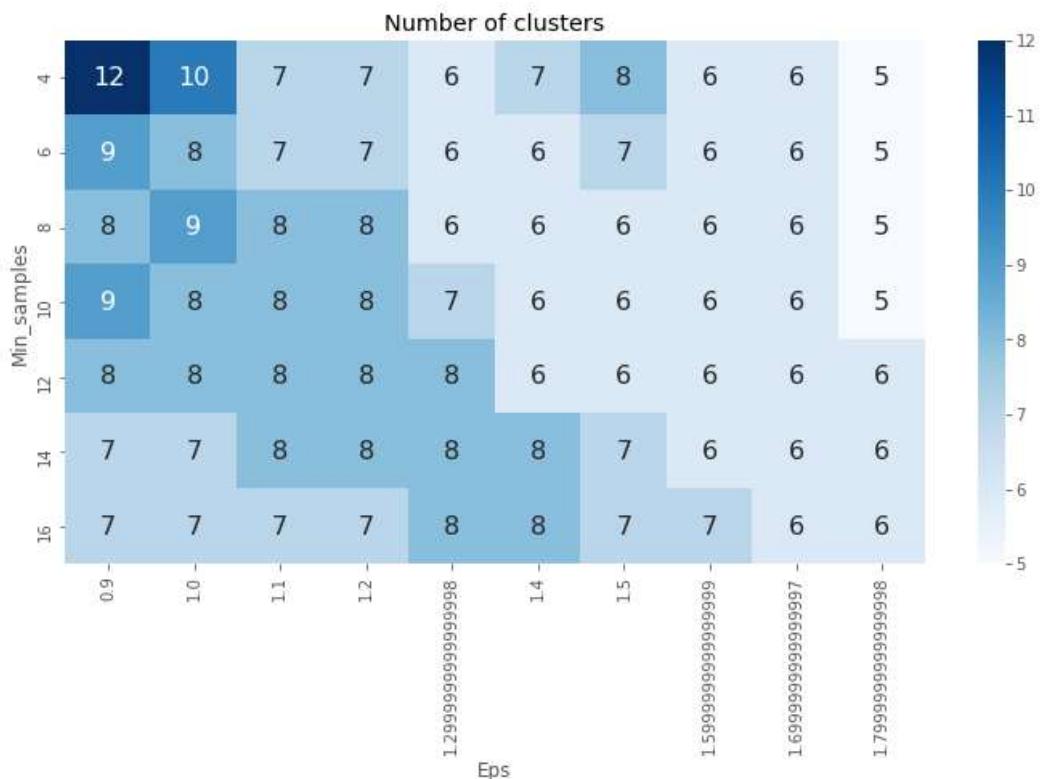
```
In [27]: #Run DBSCAN fit with multiple parameter sets to find optimal parameters for best most accurate
#Run timer to check computation time.
no_of_clusters = []
sil_score = []
print('PCU utilization:', psutil.cpu_percent(), '%')
print('Clusters created at each round. Looking for 6 with high silhouette score. This will take about 200s to run')
start = time.time()

for p in DBSCAN_params:
    DBS_clustering = DBSCAN(eps=p[0], min_samples=p[1]).fit(X_data)
    no_of_clusters.append(len(np.unique(DBS_clustering.labels_)))
    #print(DBS_clustering.labels_)
    #print(no_of_clusters[-1])
    #print(p[0],p[1])
    #print('PCU utilization%:', psutil.cpu_percent())
    if len(np.unique(DBS_clustering.labels_)) > 1:
        sil_score.append(silhouette_score(X_data, DBS_clustering.labels_))
        good_label = DBS_clustering.labels_
    else:
        sil_score.append(silhouette_score(X_data, good_label))

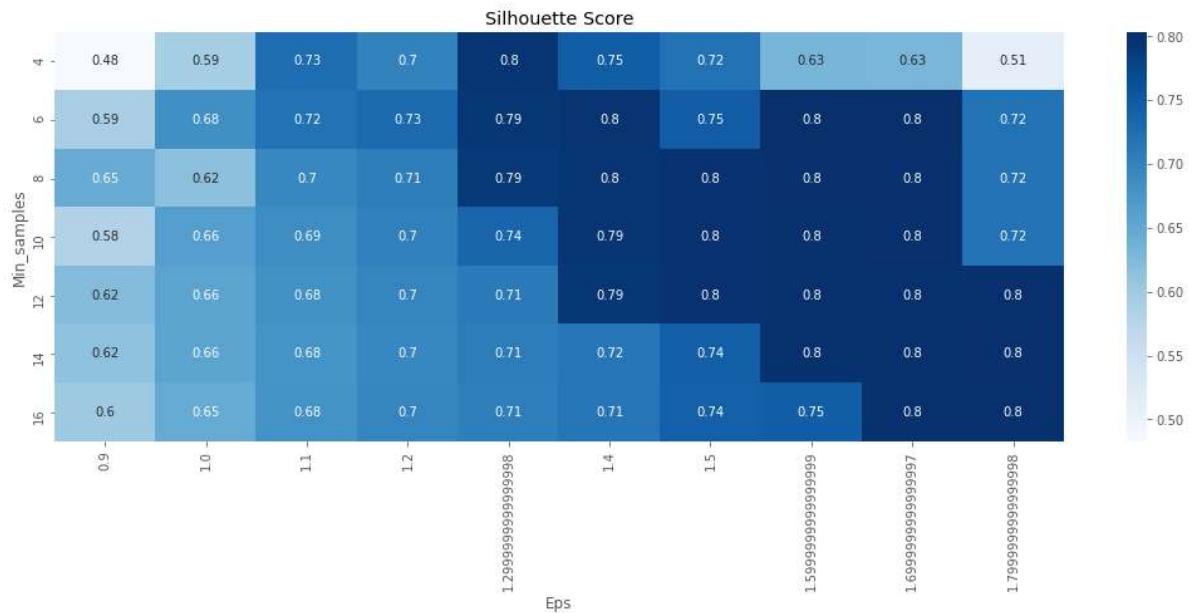
end = time.time()
print('PCU utilization:', psutil.cpu_percent(), '%')
print('Total time: %0.2f'%(end-start), 's')
```

PCU utilization: 27.5 %
Clusters created at each round. Looking for 6 with high silhouette score. This will take about 200s to run
PCU utilization: 53.6 %
Total time: 136.70 s

```
In [28]: #Show clustering matrix with number of clusters, eps and min_samples (x,y).
import matplotlib.ticker as ticker
tmp = pd.DataFrame.from_records(DBSCAN_params, columns=['Eps', 'Min_samples'])
tmp['No_of_clusters'] = no_of_clusters
pivot_1 = pd.pivot_table(tmp, values='No_of_clusters', index='Min_samples', columns='Eps')
fig, ax = plt.subplots(figsize=(12,6))
sns.heatmap(pivot_1, annot=True, annot_kws={"size": 16}, cmap="Blues", ax=ax)
ax.set_title('Number of clusters')
plt.show()
```



```
In [29]: 1 ##Show clustering matrix with silhouette score, eps and min_samples (x,y).
2 #Silhouette score indicates how well clusters are separates. Range is -1..1. 1 is the best score
3 from matplotlib.ticker import (MultipleLocator,
4                                 FormatStrFormatter,
5                                 AutoMinorLocator)
6 tmp = pd.DataFrame.from_records(DBSCAN_params, columns=['Eps', 'Min_samples'])
7 tmp['Sil_score'] = sil_score
8
9 pivot_1 = pd.pivot_table(tmp, values='Sil_score', index='Min_samples', columns='Eps')
10
11 fig, ax = plt.subplots(figsize=(18,6))
12 sns.heatmap(pivot_1, annot=True, annot_kws={"size": 10}, cmap="Blues", ax=ax)
13 ax.set_title('Silhouette Score')
14 plt.show()
```



```
In [30]: 1 #Final clustering with most optimal eps and min_samples parameters selected from the above.
2 print('PCU utilization:', psutil.cpu_percent(), '%')
3 start = time.time()
4
5 DBS_clustering = DBSCAN(eps=1.6, min_samples=12).fit(X_data)#1.6&10 goood!
6 core_samples_mask = np.zeros_like(DBS_clustering.labels_, dtype=bool)
7 core_samples_mask[DBS_clustering.core_sample_indices_] = True
8 labels = DBS_clustering.labels_
9 n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
10 n_noise_ = list(labels).count(-1)
11 DBSCAN_clustered = pd.DataFrame(X_data.copy())
12 DBSCAN_clustered.loc[:, 'Cluster'] = DBS_clustering.labels_ # append labels to points
13
14 end = time.time()
15 print('PCU utilization:', psutil.cpu_percent(), '%')
16 print('Total time: %0.2f'%(end-start), 's')
```

PCU utilization: 20.5 %
 PCU utilization: 42.5 %
 Total time: 0.46 s

```
In [31]: 1 #Check number of clusters and amount of samples in each.  
2 DBSCAN_clust_sizes = DBSCAN_clustered.groupby('Cluster').size().to_frame()  
3 DBSCAN_clust_sizes.columns = ["DBSCAN_size"]  
4 DBSCAN_clust_sizes
```

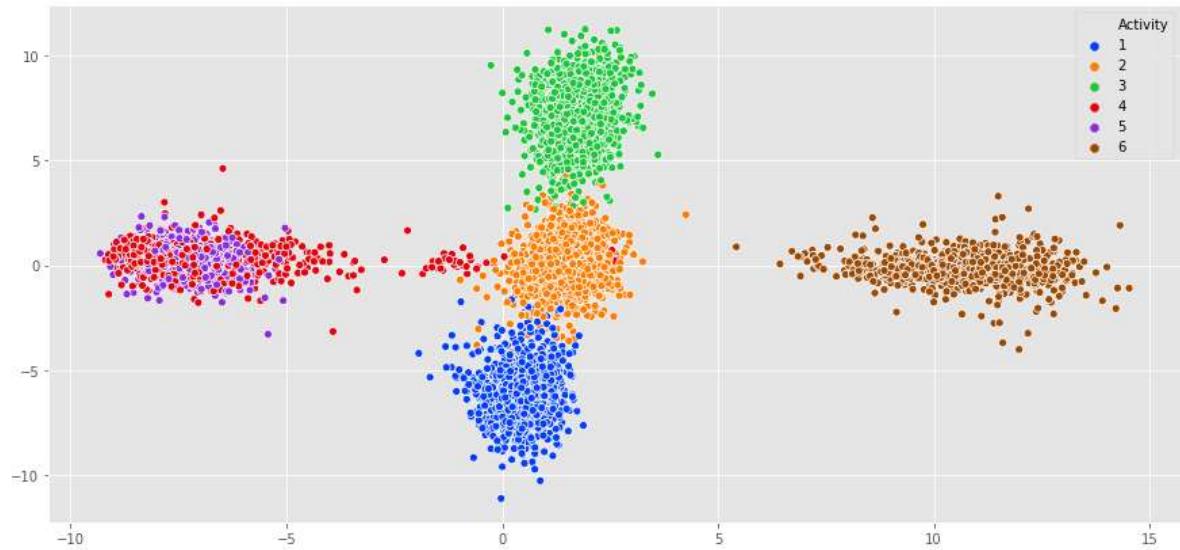
Out[31]:

DBSCAN_size

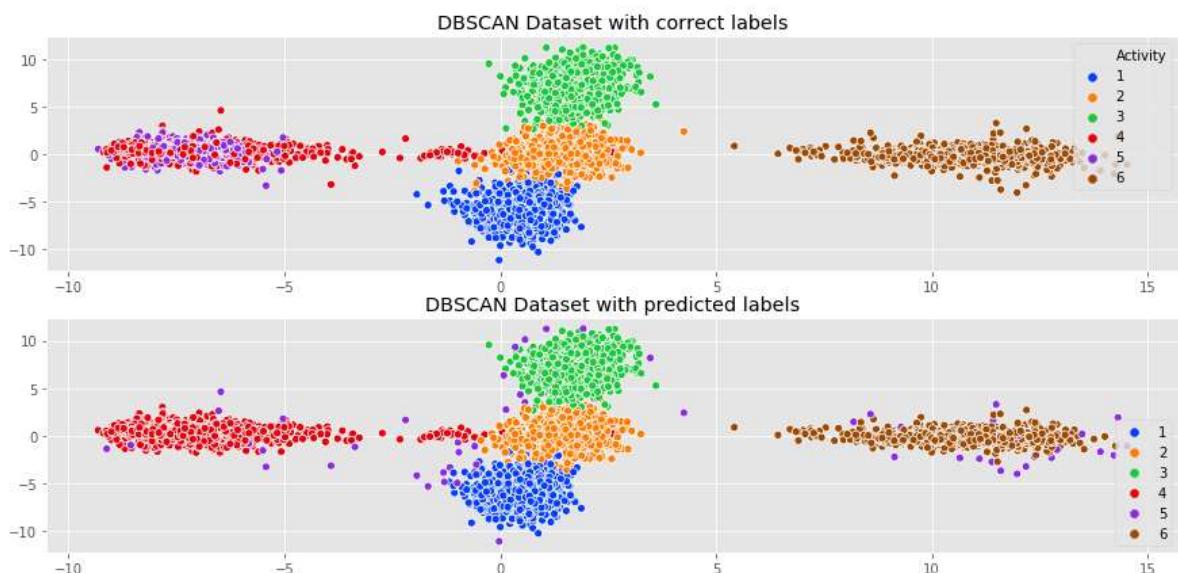
Cluster	DBSCAN_size
-1	81
0	2643
1	1377
2	1216
3	972
4	1063

```
In [32]: 1 #Plot the original cluster assignments using given cluster assignments.  
2 sns.scatterplot(X_data[:,1], X_data[:,2], hue=train_data["Activity"], palette="bright")
```

Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x231ab81c548>



```
In [33]: #Plot DBSCAN predicted clusters. Label '5' is outliers. Sitting and standing in same cluster
#Match the colors with original.
DBSCAN_clustered['Cluster'] = DBSCAN_clustered['Cluster'] + 10
#Relabeling to match input data
data_labels = DBSCAN_clustered.Cluster.values
data_labels[data_labels == 10] = 4
data_labels[data_labels == 11] = 6
data_labels[data_labels == 12] = 1
data_labels[data_labels == 13] = 3
data_labels[data_labels == 14] = 2
data_labels[data_labels == 9] = 5
plt.subplot(211)
sns.scatterplot(X_data[:,1], X_data[:,2], hue=train_data["Activity"], palette="bright")
plt.title('DBSCAN Dataset with correct labels')
plt.subplot(212)
sns.scatterplot(X_data[:,1], X_data[:,2], data_labels, palette="bright")
plt.title('DBSCAN Dataset with predicted labels')
plt.show()
#sns.scatterplot(X_data[:,1], X_data[:,2], data_labels, palette="bright")
```



DBSCAN KPI creation.

```
In [34]: #Print the DBSCAN KPIs.
#Reshape original label arrays to match DBSCAN labeling
label_opt = np.array(np.reshape(label,-1))
label_opt = np.array(np.reshape(label_opt,-1))
print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)
print("Homogeneity: %0.3f" % metrics.homogeneity_score(label_opt, labels))
print("Completeness: %0.3f" % metrics.completeness_score(label_opt, labels))
print("V-measure: %0.3f" % metrics.v_measure_score(label_opt, labels))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(label_opt, labels))
print("Adjusted Mutual Information: %0.3f" % metrics.adjusted_mutual_info_score(label_opt, la
12 print("Silhouette Coefficient: %0.3f" % metrics.silhouette_score(X_data, labels))
```

Estimated number of clusters: 5
 Estimated number of noise points: 81
 Homogeneity: 0.850
 Completeness: 0.962
 V-measure: 0.903
 Adjusted Rand Index: 0.787
 Adjusted Mutual Information: 0.903
 Silhouette Coefficient: 0.800

```
In [35]: 1 #Model accuracy. Cluster assignment matching the original label.
2 results1 = np.array(train_data["Activity"]) == data_labels
3 sum(bool(x) for x in results1)/len(data_labels)
```

Out[35]: 0.8039989118607181

Network Layout

```
In [36]: 1 DBS_clustering.get_params(deep=True)
```

```
Out[36]: {'algorithm': 'auto',
'eps': 1.6,
'leaf_size': 30,
'metric': 'euclidean',
'metric_params': None,
'min_samples': 12,
'n_jobs': None,
'p': None}
```

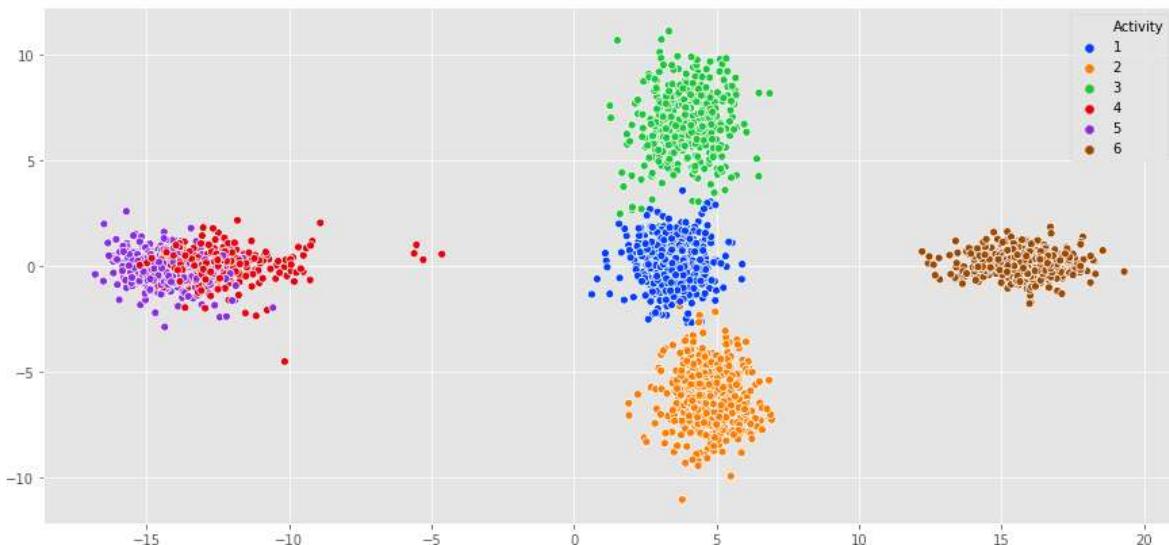
Use test data to verify model performance.

```
In [37]: 1 #Read the data. 'train_data' includes features and Label.
2 test_data = pd.read_csv("Test_Dataset.csv")
3 Xt = np.array(pd.read_csv("X_test.csv", header=None))
4 t_label = pd.read_csv("y_test.csv", header=None)
```

```
In [38]: 1 lda = LDA(n_components=4)
2 X_test = lda.fit_transform(Xt, t_label)
3 sns.scatterplot(X_test[:,1], X_test[:,3], hue=test_data['Activity'], palette='bright')
```

C:\Users\marpulli\Anaconda3\lib\site-packages\sklearn\utils\validation.py:760: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
y = column_or_1d(y, warn=True)

Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x231b2a3dbc8>



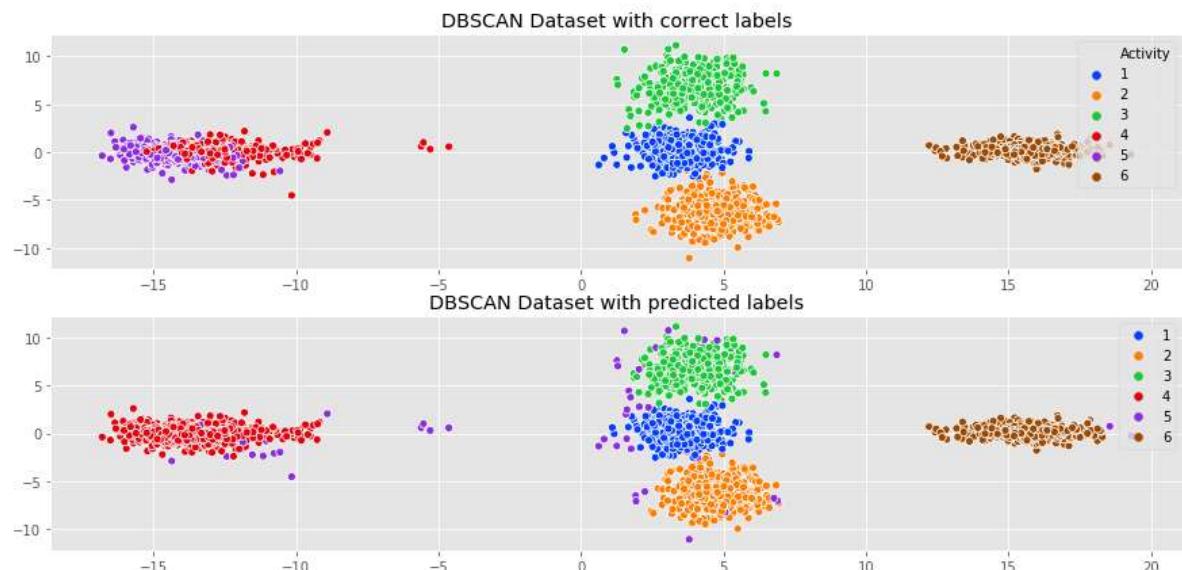
```
In [39]: 1 #Final clustering with most optimal eps and min_samples parameters selected from the above.
2 X_t_data = X_test #No scaling
3 DBS_clustering = DBSCAN(eps=1.6, min_samples=12).fit(X_t_data)
4 core_samples_mask = np.zeros_like(DBS_clustering.labels_, dtype=bool)
5 core_samples_mask[DBS_clustering.core_sample_indices_] = True
6 labels = DBS_clustering.labels_
7 n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
8 n_noise_ = list(labels).count(-1)
9 DBSCAN_clustered = pd.DataFrame(X_t_data.copy())
10 DBSCAN_clustered.loc[:, 'Cluster'] = DBS_clustering.labels_ # append labels to points
```

```
In [40]: 1 DBSCAN_clust_sizes = DBSCAN_clustered.groupby('Cluster').size().to_frame()
2 DBSCAN_clust_sizes.columns = ["DBSCAN_size"]
3 DBSCAN_clust_sizes
```

Out[40]:

Cluster	DBSCAN_size
-1	63
0	1001
1	534
2	487
3	399
4	463

```
In [41]: 1 #Plot DBSCAN predicted clusters. Label '5' is outliers. Sitting and standing in same cluster
2 #Match the colors with original.
3 DBSCAN_clustered['Cluster'] = DBSCAN_clustered['Cluster'] + 10
4 #Relabeling to match input data. Unsupervised Learning clustering done correctly but labels n
5 data_labels = DBSCAN_clustered.Cluster.values
6 data_labels[data_labels == 10] = 4
7 data_labels[data_labels == 11] = 6
8 data_labels[data_labels == 12] = 1
9 data_labels[data_labels == 13] = 3
10 data_labels[data_labels == 14] = 2
11 data_labels[data_labels == 9] = 5
12 #sns.scatterplot(X_t_data[:,1], X_t_data[:,3], data_labels, palette="bright")
13
14 plt.subplot(211)
15 sns.scatterplot(X_t_data[:,1], X_t_data[:,3], test_data["Activity"], palette="bright")
16 #sns.scatterplot(X_t_data[:,1], X_t_data[:,3], hue=test_data["Activity"], palette="bright")
17 plt.title('DBSCAN Dataset with correct labels')
18
19 plt.subplot(212)
20 #sns.scatterplot(X_t_data[:,1], X_t_data[:,3], data_labels, palette="bright")
21 sns.scatterplot(X_t_data[:,1], X_t_data[:,3], data_labels, palette="bright")
22 plt.title('DBSCAN Dataset with predicted labels')
23
24 plt.show()
```



```
In [42]: 1 #Print clustering KPIs
2 label_opt = np.array(np.reshape(t_label,-1))
3 label_opt = np.array(np.reshape(label_opt,-1))
4 print('Estimated number of clusters: %d' % n_clusters_)
5 print('Estimated number of noise points: %d' % n_noise_)
6 print("Homogeneity: %0.3f" % metrics.homogeneity_score(label_opt, labels))
7 print("Completeness: %0.3f" % metrics.completeness_score(label_opt, labels))
8 print("V-measure: %0.3f" % metrics.v_measure_score(label_opt, labels))
9 print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(label_opt, labels))
10 print("Adjusted Mutual Information: %0.3f" % metrics.adjusted_mutual_info_score(label_opt, la
11 print("Silhouette Coefficient: %0.3f" % metrics.silhouette_score(X_t_data, labels))
```

Estimated number of clusters: 5
 Estimated number of noise points: 63
 Homogeneity: 0.849
 Completeness: 0.938
 V-measure: 0.891
 Adjusted Rand Index: 0.793
 Adjusted Mutual Information: 0.891
 Silhouette Coefficient: 0.825

```
In [43]: 1 #Model accuracy. Cluster assignment matching the original label. Test accuracy is very close
2 results1 = np.array(test_data["Activity"] == data_labels
3 sum(bool(x) for x in results1)/len(data_labels)
```

Out[43]: 0.8048863250763488

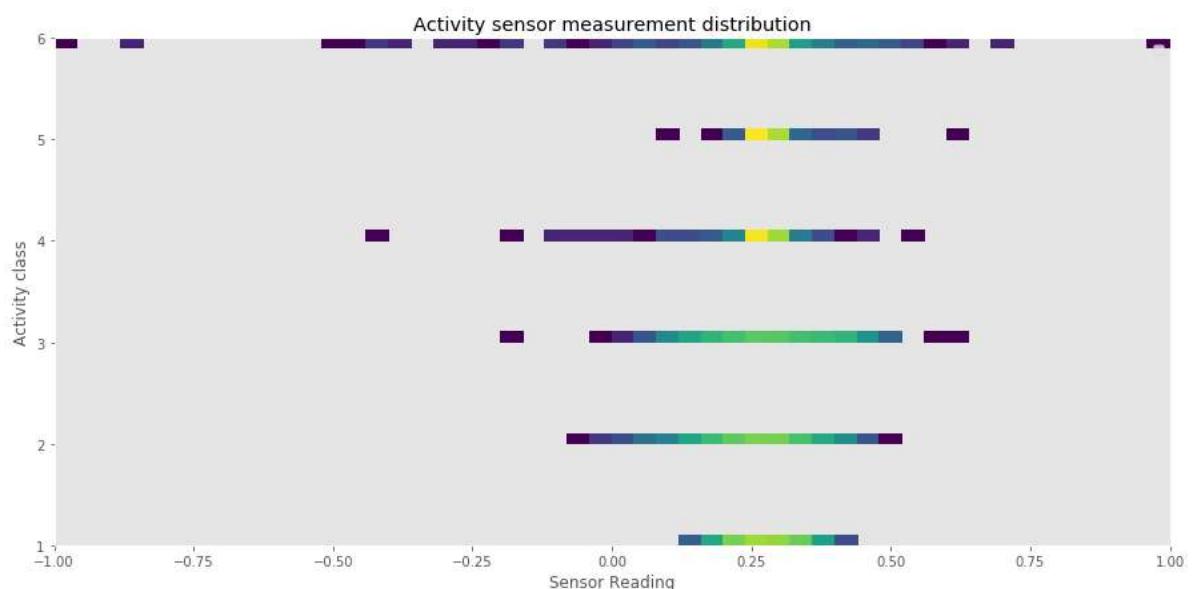
GMM clustering.

```
In [44]: 1 from sklearn.cluster import KMeans
2 from sklearn.mixture import GaussianMixture
3 from matplotlib import colors
```

```
In [45]: 1 #Let's check the data distribution. Follows normal distribution.
2 plt.hist2d(train_data['1 tBodyAcc-mean()-X'], train_data['Activity'], bins=(50,50), norm=colors
3 plt.xlabel('Sensor Reading')
4 plt.ylabel('Activity class')
5 plt.title('Activity sensor measurement distribution')
6 plt.legend(loc='upper right')
```

No handles with labels found to put in legend.

Out[45]: <matplotlib.legend.Legend at 0x231b2587bc8>

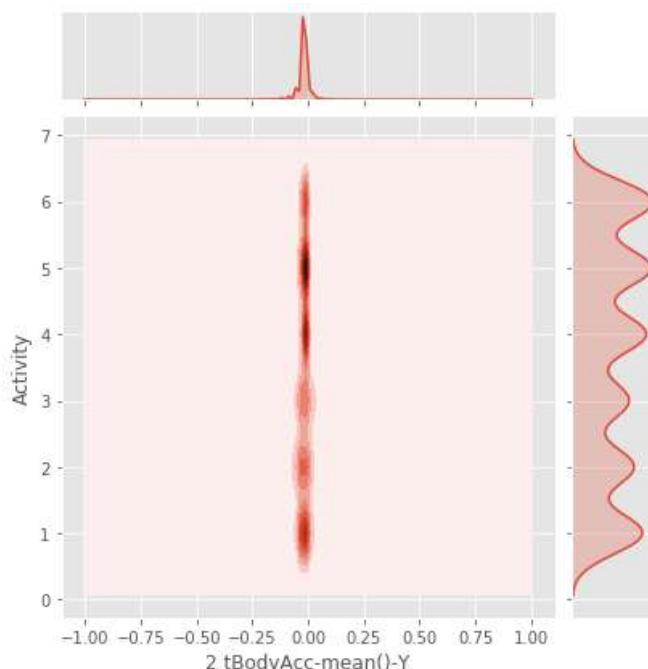
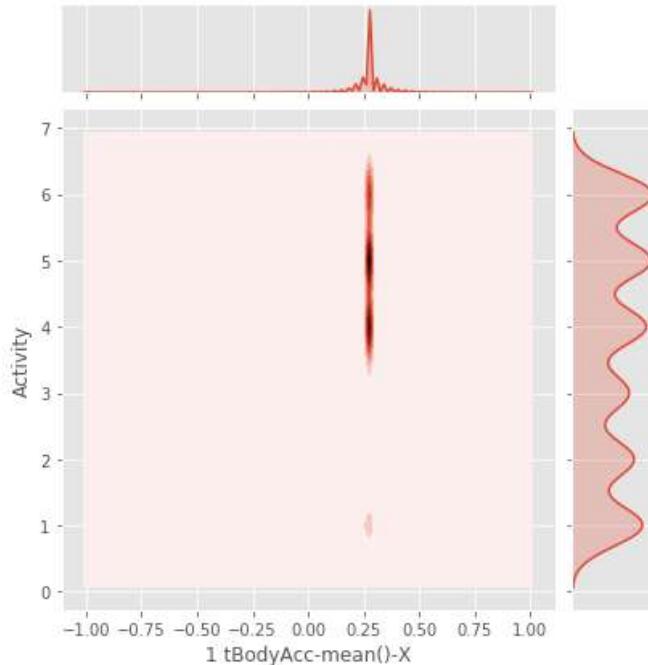


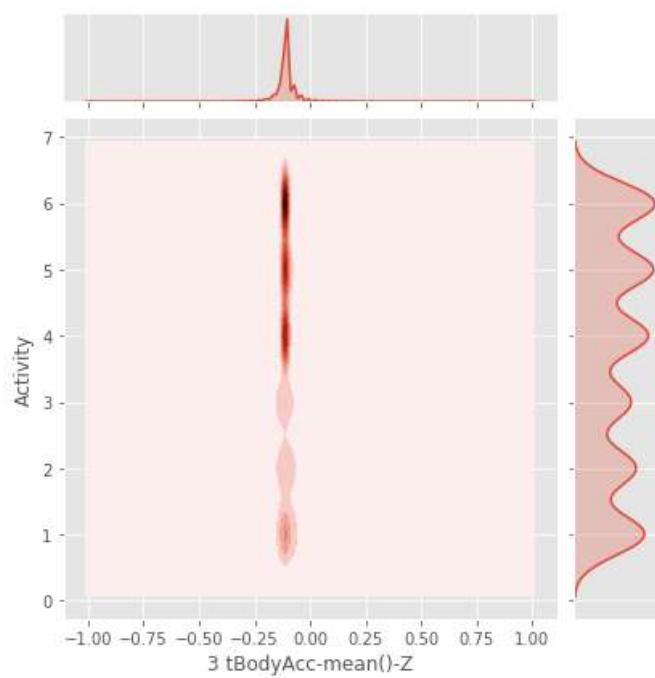
In [46]:

```
1 #Other way to check data distribution. As seen data follows normal distribution.
2 sns.jointplot(data=train_data, x="1 tBodyAcc-mean()-X", y="Activity", hue="Activity", kind="k"
3 sns.jointplot(data=train_data, x="2 tBodyAcc-mean()-Y", y="Activity", hue="Activity", kind="k"
4 sns.jointplot(data=train_data, x="3 tBodyAcc-mean()-Z", y="Activity", hue="Activity", kind="k

C:\Users\marpulli\Anaconda3\lib\site-packages\seaborn\distributions.py:437: UserWarning: The fol
lowing kwargs were not used by contour: 'hue'
    cset = contour_func(xx, yy, z, n_levels, **kwargs)
C:\Users\marpulli\Anaconda3\lib\site-packages\seaborn\distributions.py:437: UserWarning: The fol
lowing kwargs were not used by contour: 'hue'
    cset = contour_func(xx, yy, z, n_levels, **kwargs)
C:\Users\marpulli\Anaconda3\lib\site-packages\seaborn\distributions.py:437: UserWarning: The fol
lowing kwargs were not used by contour: 'hue'
    cset = contour_func(xx, yy, z, n_levels, **kwargs)
```

Out[46]: <seaborn.axisgrid.JointGrid at 0x231b25dce08>





In [48]:

```

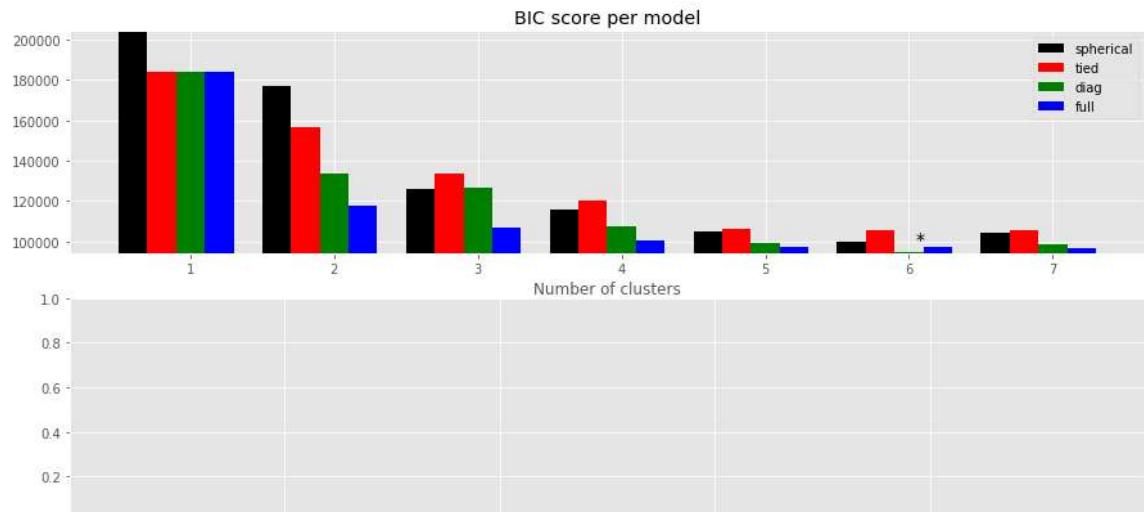
1 #from scikit_learn
2
3 print(__doc__)
4
5 import itertools
6 import matplotlib as mpl
7 from scipy import linalg
8 from sklearn import mixture
9
10 # Number of samples per component
11 n_samples = 400
12
13 # Generate random sample, two components
14 #np.random.seed(0)
15 #C = np.array([[0., -0.1], [1.7, .4]])
16 X = X_data
17
18 lowest_bic = np.infty
19 bic = []
20 n_components_range = range(1, 8)
21 cv_types = ['spherical', 'tied', 'diag', 'full']
22 for cv_type in cv_types:
23     for n_components in n_components_range:
24         # Fit a mixture of Gaussians with EM
25         gmm = GaussianMixture(n_components=n_components, covariance_type=cv_type)
26         gmm.fit(X)
27         bic.append(gmm.bic(X))
28         if bic[-1] < lowest_bic:
29             lowest_bic = bic[-1]
30             best_gmm = gmm
31
32 bic = np.array(bic)
33 color_iter = itertools.cycle(['k', 'r', 'g', 'b', 'c', 'm', 'y'])
34 clf = best_gmm
35 bars = []
36
37 # Plot the BIC scores
38 spl = plt.subplot(2, 1, 1)
39 for i, (cv_type, color) in enumerate(zip(cv_types, color_iter)):
40     xpos = np.array(n_components_range) + .2 * (i - 2)
41     bars.append(plt.bar(xpos, bic[i * len(n_components_range):
42                                     (i + 1) * len(n_components_range)],
43                         width=.2, color=color))
44 plt.xticks(n_components_range)
45 plt.ylim([bic.min() * 1.01 - .01 * bic.max(), bic.max()])
46 plt.title('BIC score per model')
47 xpos = np.mod(bic.argmax(), len(n_components_range)) + .65 + \
48     .2 * np.floor(bic.argmax() / len(n_components_range))
49 plt.text(xpos, bic.min() * 0.97 + .03 * bic.max(), '*', fontsize=14)
50 spl.set_xlabel('Number of clusters')
51 spl.legend([b[0] for b in bars], cv_types)
52
53 # Plot the winner
54 splot = plt.subplot(2, 1, 2)
55 Y_ = clf.predict(X)
56 for i, (mean, covar, color) in enumerate(zip(clf.means_, clf.covariances_,
57                                               color_iter)):
58     v, w = linalg.eigh(covar)
59     #print(v)
60     #print(w)
61     if not np.any(Y_ == i):
62         continue
63     plt.scatter(X[Y_ == i, 0], X[Y_ == i, 1], .8, color=color)
64
65     # Plot an ellipse to show the Gaussian component
66     angle = np.arctan2(w[0][1], w[0][0])
67     angle = 180 * angle / np.pi # convert to degrees
68     v *= 4
69     ell = mpl.patches.Ellipse(mean, v[0], v[1], 180 + angle, color=color)
70     ell.set_clip_box(splot.bbox)
71     ell.set_alpha(.5)
72     splot.add_artist(ell)
73
74 #plt.xlim(-10, 10)
75 #plt.ylim(-3, 6)
76 #plt.xticks(())

```

```

77 #plt.yticks(())
78 plt.title('Selected GMM: full model, 6 components')
79 plt.subplots_adjust(hspace=.35, bottom=.02)
80 #plt.show()

```



In [49]:

```

1 #Create model and fit it.
2 gmm=GaussianMixture(n_components=6, covariance_type='full', max_iter = 600, random_state = 3)
3 gmm.fit(X_data, label)

```

Out[49]:

```
GaussianMixture(covariance_type='full', init_params='kmeans', max_iter=600,
                 means_init=None, n_components=6, n_init=1, precisions_init=None,
                 random_state=3, reg_covar=1e-06, tol=0.001, verbose=0,
                 verbose_interval=10, warm_start=False, weights_init=None)
```

In [50]:

```

1 #This is Log-Likelihood score. This does not have much meaning regarding the accuracy except
2 gmm.score(X_data)

```

Out[50]:

```
-6.550598356220418
```

In [51]:

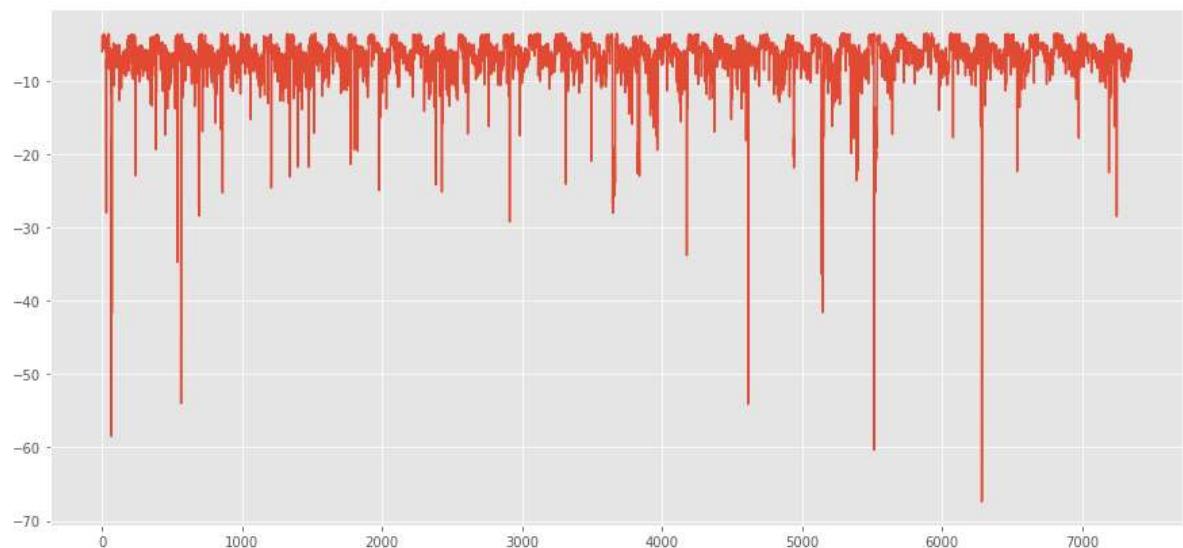
```

1 #Make prediction. Graph is Log likelihood.
2 predG=gmm.predict(X_data)
3 score_S = gmm.score_samples(X_data)
4 plt.plot(score_S)

```

Out[51]:

```
[<matplotlib.lines.Line2D at 0x231a97cbac8>]
```



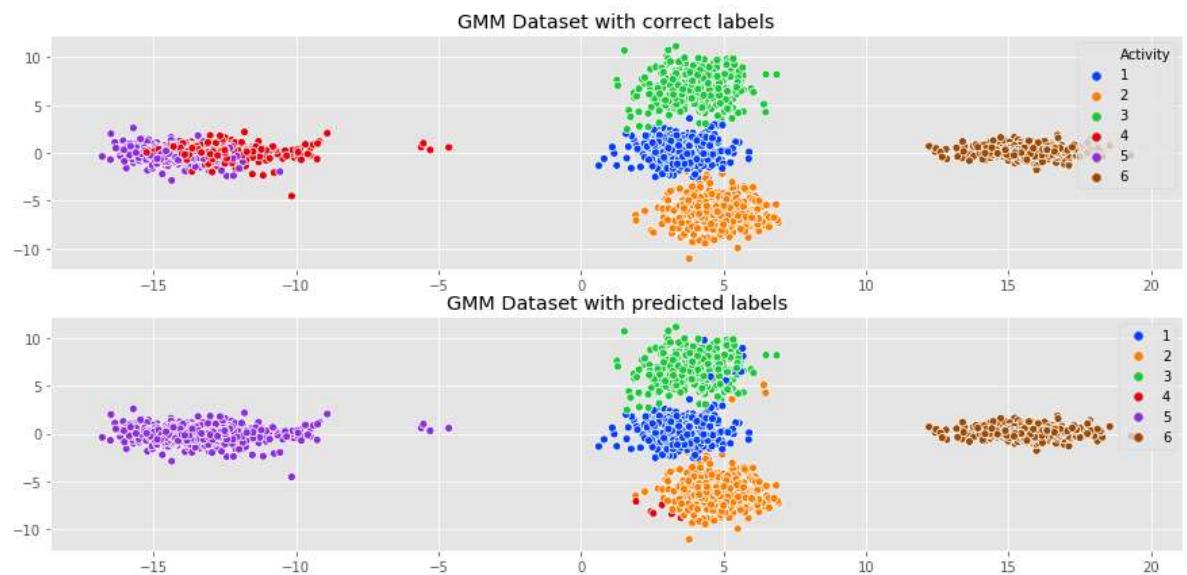
```
In [ ]: ► 1 #Plot train clusters with given labels.  
2 #sns.scatterplot(X_train[:,2], X_train[:,0], hue=predG, palette='bright')  
3 fig, ax = plt.subplots()  
4 sns.scatterplot(X_data[:,1], X_data[:,3], hue=train_data['Activity'], palette='bright')  
5 #Zoom in to 3&5 border.  
6 ax2 = plt.axes([0.12, -0.2, .3, .2], facecolor='lightgrey')  
7 sns.scatterplot(X_data[:,2], X_data[:,0], hue=train_data['Activity'], palette='bright')  
8 ax2.set_title('Zoom in clusters 3 & 5 border.')  
9 ax2.set_xlim([6,7.75])  
10 ax2.set_ylim([-20,-10])  
11 ax2.legend(loc='upper right')
```

```
In [ ]: ► 1 #Plot clusters with predicted labels.  
2 #sns.scatterplot(X_train[:,2], X_train[:,0], hue=predG, palette='bright')  
3 fig, ax = plt.subplots()  
4 sns.scatterplot(X_data[:,1], X_data[:,3], hue=predG, palette='bright')  
5 #Zoom in to 3&5 border.  
6 ax2 = plt.axes([0.12, -0.2, .3, .2], facecolor='lightgrey')  
7 sns.scatterplot(X_data[:,1], X_data[:,3], hue=predG, palette='bright')  
8 ax2.set_title('Zoom in clusters 3 & 5 border.')  
9 ax2.set_xlim([6,7.75])  
10 ax2.set_ylim([-20,-10])  
11 ax2.legend(loc='upper right')
```

```
In [ ]: ► 1 #Plot DBSCAN predicted clusters. Label '5' is outliers. Sitting and standing in same cluster  
2 #Match the colors with original.  
3 predG10=[]  
4 predG10 = predG + 10  
5 #Relabeling to match input data  
6 data_labels = predG  
7 data_labels[data_labels == 10] = 5  
8 data_labels[data_labels == 11] = 2  
9 data_labels[data_labels == 12] = 6  
10 data_labels[data_labels == 13] = 3  
11 data_labels[data_labels == 14] = 1  
12 data_labels[data_labels == 15] = 4  
13 #sns.scatterplot(X_data[:,1], X_data[:,3], data_labels, palette="bright")
```

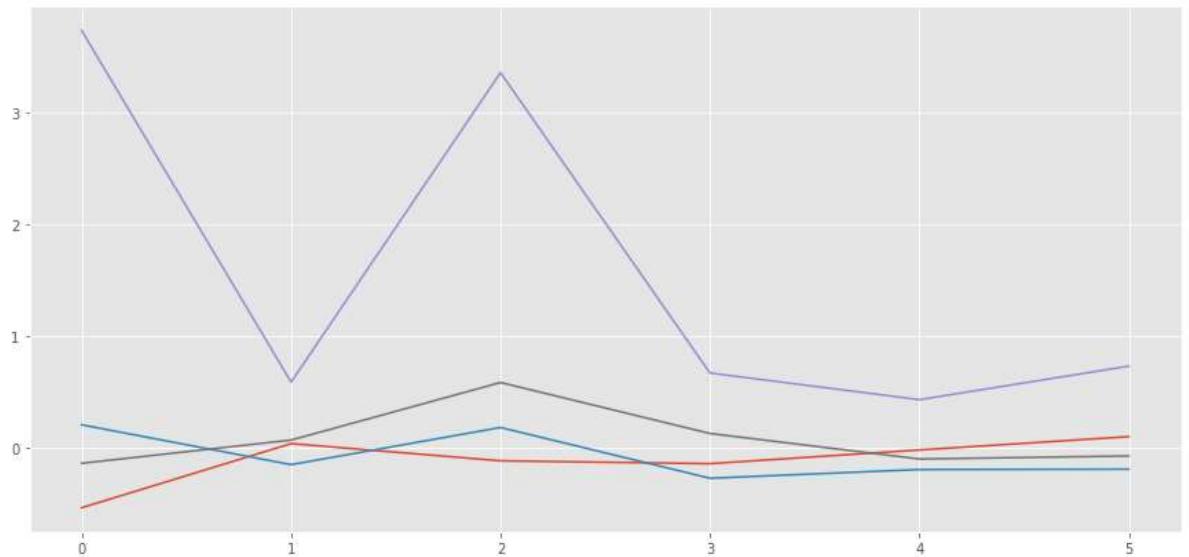
In [52]:

```
1 #Plot test data clusters and predicted data clusters.
2 gmm_pred = gmm.predict(X_t_data)
3
4 #Map cluster labels.
5 gmm_pred10 = []
6 gmm_pred10 = gmm_pred + 10
7 #Relabeling to match input data
8 data_labels = gmm_pred10
9 data_labels[data_labels == 10] = 5
10 data_labels[data_labels == 11] = 2
11 data_labels[data_labels == 12] = 6
12 data_labels[data_labels == 13] = 3
13 data_labels[data_labels == 14] = 1
14 data_labels[data_labels == 15] = 4
15
16 plt.subplot(211)
17 sns.scatterplot(X_t_data[:,1], X_t_data[:,3], test_data["Activity"], palette="bright")
18 plt.title('GMM Dataset with correct labels')
19
20 plt.subplot(212)
21 sns.scatterplot(X_t_data[:,1], X_t_data[:,3], data_labels, palette="bright")
22 plt.title('GMM Dataset with predicted labels')
23
24 plt.show()
```



```
In [53]: #The precision matrices for each component in the mixture. A precision matrix is the inverse
#Just one component shown.
plt.plot(gmm.precision_[ :, 2])
```

```
Out[53]: [
```



```
In [54]: #Accuracy of model. Cluster assignment matching the original label. True positives (TP).
results1 = np.array(test_data["Activity"]) == data_labels
sum(bool(x) for x in results1)/len(data_labels)
```

```
Out[54]: 0.8262639972853749
```