Introduction to Data Science

Project 2

Seonhye Yang

1.1.The missing values in “movieReplicationSet.csv” were replaced with mean of each column using a loop. To find the most correlated user for every user in the given data frame, the data frame was transposed. However, the correlation data frame’s diagonal has values 1’s. If we were to find the largest correlation value per user, the results will be all 1’s. To avoid this issue, the correlation diagonal values were replaced with 0 so the diagonal values don’t interfere when trying to find the maximum value.

{0: 314, 1: 896, 2: 831, 3: 704, 4: 660, 5: 574, 6: 521, 7: 850, 8: 412, 9: 966, 10: 831, 11: 831, 12: 896, 13: 896,
14: 265, 15: 896, 16: 515, 17: 831, 18: 896, 19: 37, 20: 832, 21: 497, 22: 409, 23: 896, 24: 86, 25: 409, 26: 825, 2
7: 831, 28: 896, 29: 896, 30: 831, 31: 272, 32: 896, 33: 240, 34: 896, 35: 594, 36: 66, 37: 831, 38: 896, 39: 831, 4
0: 896, 41: 896, 42: 831, 43: 1029, 44: 239, 45: 328, 46: 239, 47: 452, 48: 300, 49: 825, 50: 831, 51: 831, 52: 831,
53: 660, 54: 27, 55: 896, 56: 1051, 57: 896, 58: 896, 59: 896, 60: 488, 61: 452, 62: 250, 63: 896, 64: 588, 65: 239,
66: 272, 67: 896, 68: 831, 69: 239, 70: 133, 71: 831, 72: 412, 73: 1010, 74: 1083, 75: 1031, 76: 409, 77: 674, 78: 51
6, 79: 130, 80: 521, 81: 20, 82: 897, 83: 74, 84: 17, 85: 896, 86: 452, 87: 156, 88: 896, 89: 852, 90: 896, 91: 504,
92: 25, 93: 284, 94: 831, 95: 239, 96: 831, 97: 531, 98: 453, 99: 452, 100: 896, 101: 831, 102: 939, 103: 896, 104: 5
88, 105: 113, 106: 831, 107: 896, 108: 896, 109: 831, 110: 152, 111: 896, 112: 306, 113: 896, 114: 896, 115: 831, 11
6: 1083, 117: 404, 118: 831, 119: 831, 120: 831, 121: 875, 122: 1018, 123: 896, 124: 831, 125: 351, 126: 875, 127: 83
1, 128: 831, 129: 660, 130: 831, 131: 831, 132: 831, 133: 896, 134: 896, 135: 582, 136: 896, 137: 323, 138: 896, 139:
896, 140: 896, 141: 896, 142: 292, 143: 831, 144: 774, 145: 235, 146: 239, 147: 831, 148: 398, 149: 896, 150: 710, 15
1: 239, 152: 831, 153: 896, 154: 994, 155: 152, 156: 219, 157: 907, 158: 776, 159: 116, 160: 583, 161: 958, 162: 831,
163: 25, 164: 239, 165: 896, 166: 896, 167: 896, 168: 896, 169: 896, 170: 831, 171: 831, 172: 643, 173: 896, 174: 96
3, 175: 896, 176: 831, 177: 363, 178: 896, 179: 831, 180: 793, 181: 831, 182: 831, 183: 239, 184: 896, 185: 519, 186:
896, 187: 896, 188: 853, 189: 2, 190: 265, 191: 917, 192: 642, 193: 831, 194: 504, 195: 433, 196: 472, 197: 896, 198:
671, 199: 831, 200: 467, 201: 896, 202: 896, 203: 30, 204: 896, 205: 353, 206: 710, 207: 452, 208: 758, 209: 831, 21
0: 932, 211: 166, 212: 831, 213: 896, 214: 831, 215: 968, 216: 896, 217: 831, 218: 472, 219: 831, 220: 831, 221: 709,
222: 831, 223: 831, 224: 1010, 225: 372, 226: 896, 227: 896, 228: 178, 229: 1021, 230: 21, 231: 831, 232: 896, 233: 1
32, 234: 570, 235: 831, 236: 736, 237: 896, 238: 831, 239: 831, 240: 831, 241: 49, 242: 1046, 243: 831, 244: 167, 24
5: 831, 246: 896, 247: 896, 248: 831, 249: 896, 250: 831, 251: 229, 252: 831, 253: 896, 254: 898, 255: 896, 256: 831,
257: 831, 258: 25, 259: 831, 260: 588, 261: 999, 262: 148, 263: 896, 264: 1043, 265: 494, 266: 434, 267: 143, 268: 26
9, 269: 831, 270: 951, 271: 831, 272: 831, 273: 896, 274: 703, 275: 1091, 276: 452, 277: 896, 278: 829, 279: 461, 28
0: 239, 281: 831, 282: 831, 283: 434, 284: 896, 285: 558, 286: 831, 287: 152, 288: 908, 289: 694, 290: 573, 291: 998,
292: 472, 293: 831, 294: 710, 295: 643, 296: 994, 297: 896, 298: 48, 299: 831, 300: 908, 301: 239, 302: 896, 303: 36
3, 304: 693, 305: 363, 306: 896, 307: 896, 308: 896, 309: 704, 310: 2, 311: 831, 312: 1081, 313: 393, 314: 831, 315:
896, 316: 896, 317: 896, 318: 515, 319: 831, 320: 831, 321: 831, 322: 896, 323: 720, 324: 831, 325: 831, 326: 896, 32
7: 831, 328: 896, 329: 831, 330: 896, 331: 30, 332: 972, 333: 831, 334: 230, 335: 404, 336: 831, 337: 21, 338: 831, 3
39: 831, 340: 831, 341: 896, 342: 896, 343: 990, 344: 896, 345: 831, 346: 338, 347: 248, 348: 831, 349: 896, 350: 106
9, 351: 831, 352: 978, 353: 452, 354: 831, 355: 709, 356: 831, 357: 780, 358: 595, 359: 831, 360: 831, 361: 565, 362:
896, 363: 896, 364: 831, 365: 794, 366: 831, 367: 428, 368: 589, 369: 896, 370: 46, 371: 875, 372: 831, 373: 896, 37
4: 896, 375: 212, 376: 1014, 377: 831, 378: 211, 379: 831, 380: 831, 381: 239, 382: 428, 383: 831, 384: 994, 385: 101
4, 386: 340, 387: 831, 388: 831, 389: 831, 390: 831, 391: 896, 392: 896, 393: 831, 394: 831, 395: 896, 396: 167, 397:
896, 398: 831, 399: 831, 400: 831, 401: 133, 402: 896, 403: 903, 404: 831, 405: 727, 406: 872, 407: 831, 408: 831, 40
9: 261, 410: 99, 411: 825, 412: 831, 413: 710, 414: 831, 415: 831, 416: 896, 417: 363, 418: 933, 419: 1002, 420: 831,
421: 831, 422: 156, 423: 605, 424: 896, 425: 853, 426: 213, 427: 292, 428: 831, 429: 398, 430: 679, 431: 451, 432: 36
3, 433: 831, 434: 831, 435: 896, 436: 896, 437: 412, 438: 471, 439: 831, 440: 219, 441: 896, 442: 992, 443: 239, 444:
896, 445: 896, 446: 897, 447: 908, 448: 970, 449: 831, 450: 712, 451: 831, 452: 831, 453: 831, 454: 302, 455: 932, 45
6: 831, 457: 831, 458: 452, 459: 896, 460: 831, 461: 825, 462: 831, 463: 831, 464: 831, 465: 896, 466: 896, 467: 363,
468: 896, 469: 896, 470: 831, 471: 831, 472: 896, 473: 831, 474: 272, 475: 896, 476: 588, 477: 831, 478: 896, 479: 89
6, 480: 340, 481: 825, 482: 896, 483: 896, 484: 831, 485: 831, 486: 831, 487: 392, 488: 986, 489: 831, 490: 896, 491:

1.2.To find the pairs with the highest correlation values, it’s important to take the diagonal values into consideration. Also, it’s important to not include duplicate pairs.

```
#1.2
unstack_cor = cor.unstack()
sorts = unstack_cor.sort_values(kind="quicksort")
print(sorts.drop_duplicates().idxmax())
print(sorts.drop_duplicates().max())
print("the pair of the most correlated users in the data is 896 and 831")
```

```
(896, 831)
0.998789092477981
the pair of the most correlated users in the data is 896 and 831
```

1.3.The highest correlation value is 0.998789. I did this by finding the maximum correlation using correlation data frame which returns highest correlation values per user. Then found the maximum correlation of per user. The maximum correlation is **0.998789092477981.**

1.4.In order to find the most correlated user from 0, 1,…, 9, take columns from 0 to 9 from correlation dataframe. Then take the maximum correlation value of each column. The results were as follows:

User 0: 314
User 1: 896
User 2: 831
User 3: 704
User 4: 660
User 5: 574
User 6: 521
User 7: 850
User 8: 412
User 9: 966

```
users = {}
for correlation in users_9:
    sort = users_9[correlation].sort_values(ascending=False)
    users[correlation] = sort.index[1]
print(users)
```

```
{0: 314, 1: 896, 2: 831, 3: 704, 4: 660, 5: 574, 6: 521, 7: 850, 8: 412, 9: 966}
```

2.1. To perform linear regression, it's important to make sure missing values are taken care of. Since the missing values were filled with column mean for ratings, we can do the same for the personal part of the data. Then I separated into two datasets: ratings of all users and the personal part of the data. Linear regression outputs one value, y. However, we have multiple columns for personal part of data. So, we need to create a loop that considers each column. Then I performed a linear regression on training and testing. For training model, training set average error is **0.26385416791100774**. For testing model, testing set average error is **0.6159992325212089.**

2.2. I used Ridge from sklearn to perform a ridge regression. Ridge regression is a regression method used to estimate the coefficients of multiple regression where variables are highly correlated (multi-collinearity). I then split the dataset into training (80%) and testing (20%). Then I performed ridge regression on alpha values 0.0, 1e-8, 1e-5, 0.1, 1, and 10. For training model, alpha = 0.0 gave the smallest error of **0.26385416791100763**. For testing model, alpha = 10 gave the smallest error of **0.557369217242015.**

2.3. I used LASSO regression for this question to obtain errors from training and testing. LASSO is a method of linear regression that utilizes shrinkage. Shrinkage is when values are shrunk towards some point such as the mean. Lasso also takes care of multicollinearity. I then split the dataset into training (80%) and testing (20%). I performed lasso regression using alpha values 1e-3, 1e-2, 1e-1, and 1. For training model, alpha = 0.001 gave the smallest error of **0.2719089385321419.** For testing model, alpha = 1 gave the smallest error of **0.5141212472760577.**