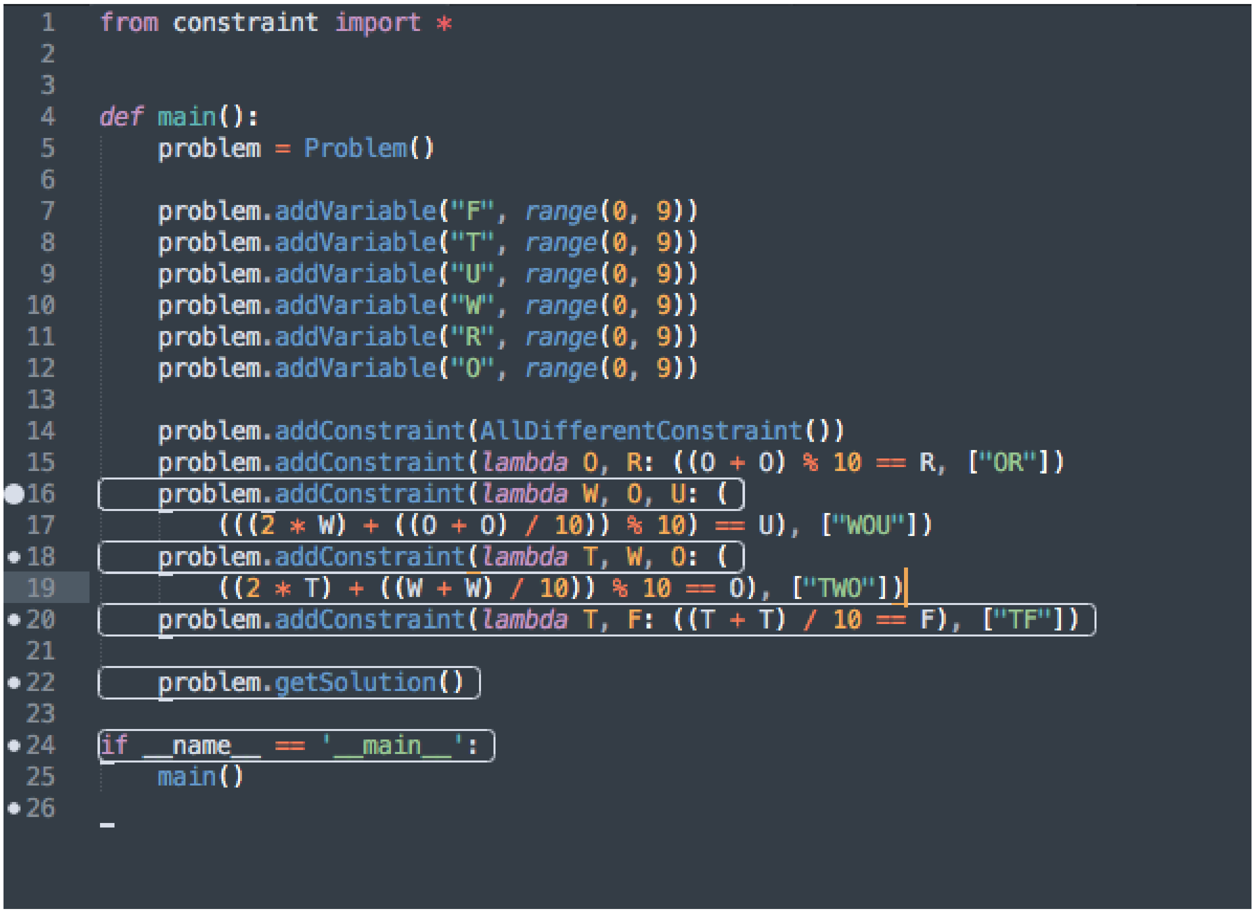
Andrew Yang
V00878595

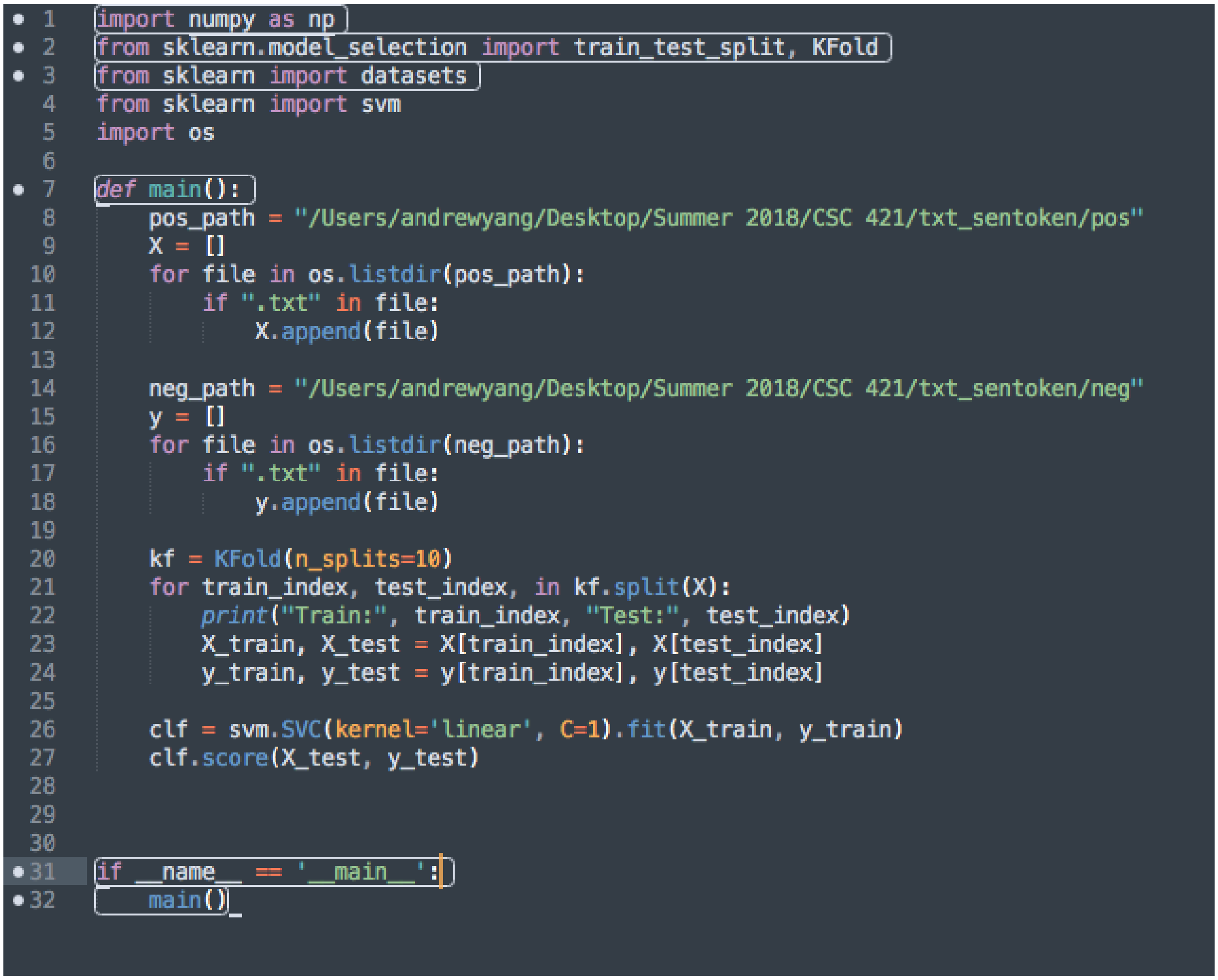
Assignment 3

Part 1:

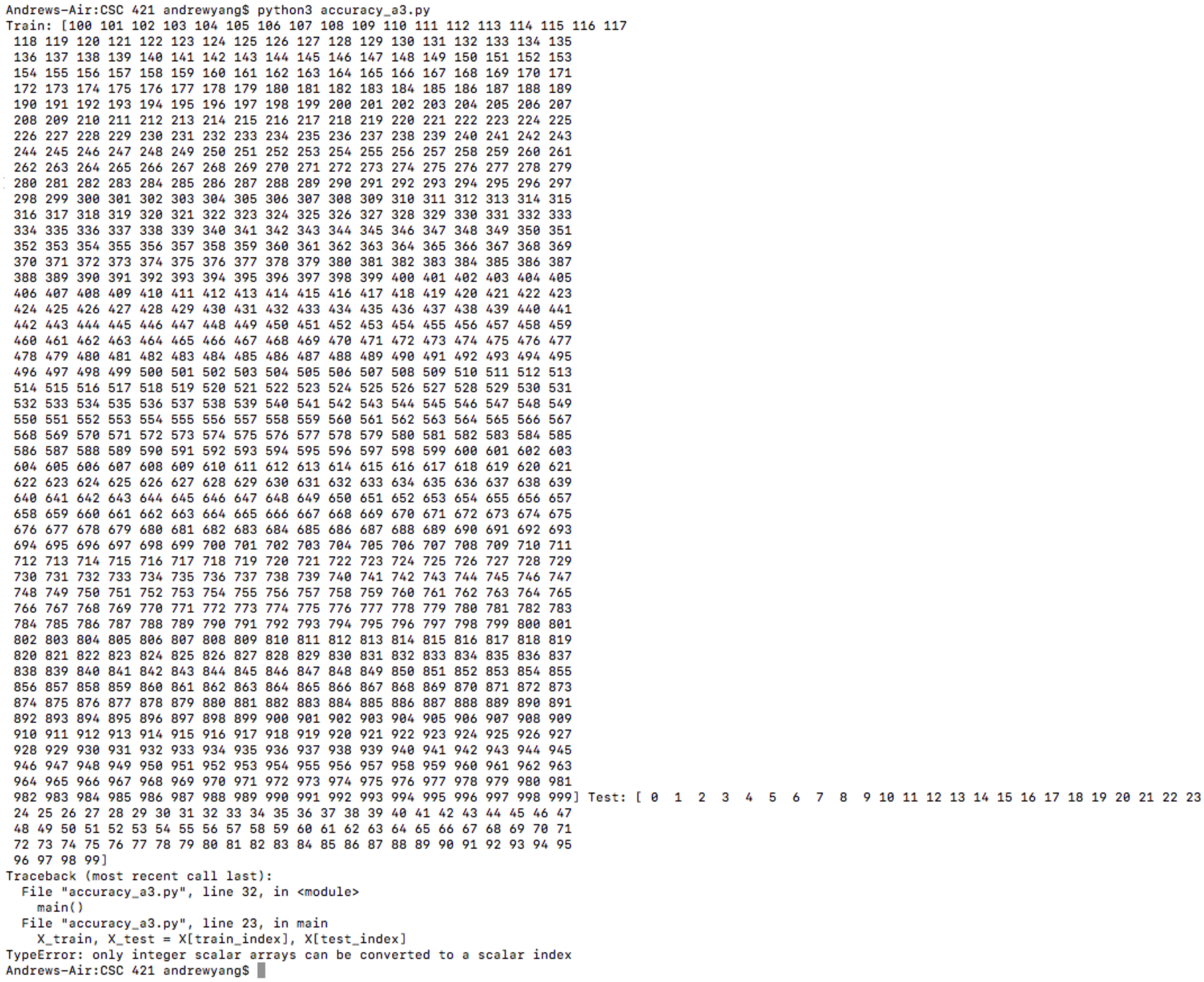


```python
from constraint import *


def main():
    problem = Problem()

    problem.addVariable("F", range(0, 9))
    problem.addVariable("T", range(0, 9))
    problem.addVariable("U", range(0, 9))
    problem.addVariable("W", range(0, 9))
    problem.addVariable("R", range(0, 9))
    problem.addVariable("O", range(0, 9))

    problem.addConstraint(AllDifferentConstraint())
    problem.addConstraint(lambda O, R: ((O + O) % 10 == R, ["OR"])
    problem.addConstraint(lambda W, O, U: (
        (((2 * W) + ((O + O) / 10)) % 10) == U), ["WOU"])
    problem.addConstraint(lambda T, W, O: (
        ((2 * T) + ((W + W) / 10)) % 10 == O), ["TWO"])
    problem.addConstraint(lambda T, F: ((T + T) / 10 == F), ["TF"])

    problem.getSolution()

if __name__ == '__main__':
    main()

```

Part 2:



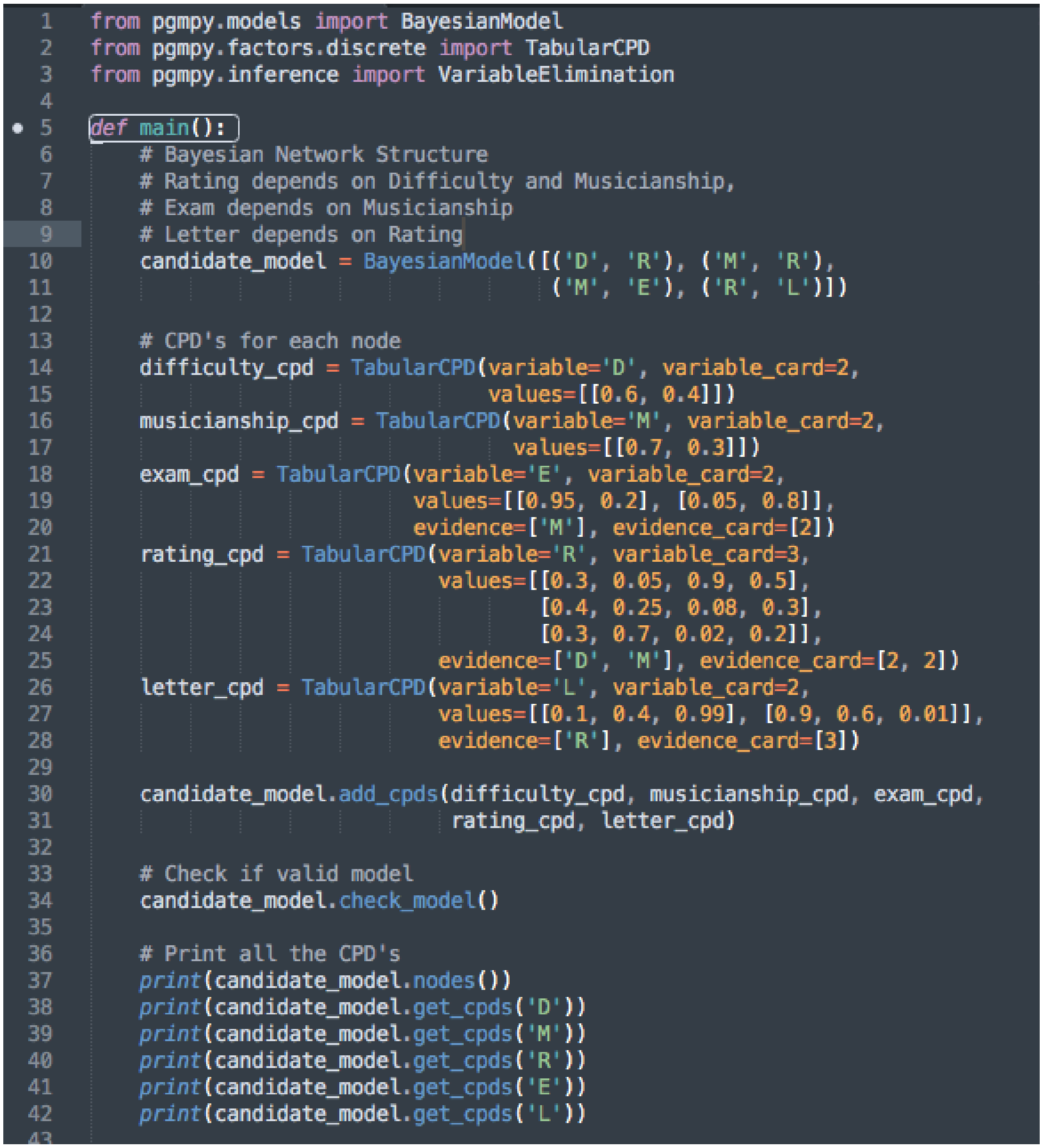
```python
import numpy as np
from sklearn.model_selection import train_test_split, KFold
from sklearn import datasets
from sklearn import svm
import os

def main():
    pos_path = "/Users/andrewyang/Desktop/Summer 2018/CSC 421/txt_sentoken/pos"
    X = []
    for file in os.listdir(pos_path):
        if ".txt" in file:
            X.append(file)

    neg_path = "/Users/andrewyang/Desktop/Summer 2018/CSC 421/txt_sentoken/neg"
    y = []
    for file in os.listdir(neg_path):
        if ".txt" in file:
            y.append(file)

    kf = KFold(n_splits=10)
    for train_index, test_index, in kf.split(X):
        print("Train:", train_index, "Test:", test_index)
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

    clf = svm.SVC(kernel='linear', C=1).fit(X_train, y_train)
    clf.score(X_test, y_test)



if __name__ == '__main__':
    main()
```



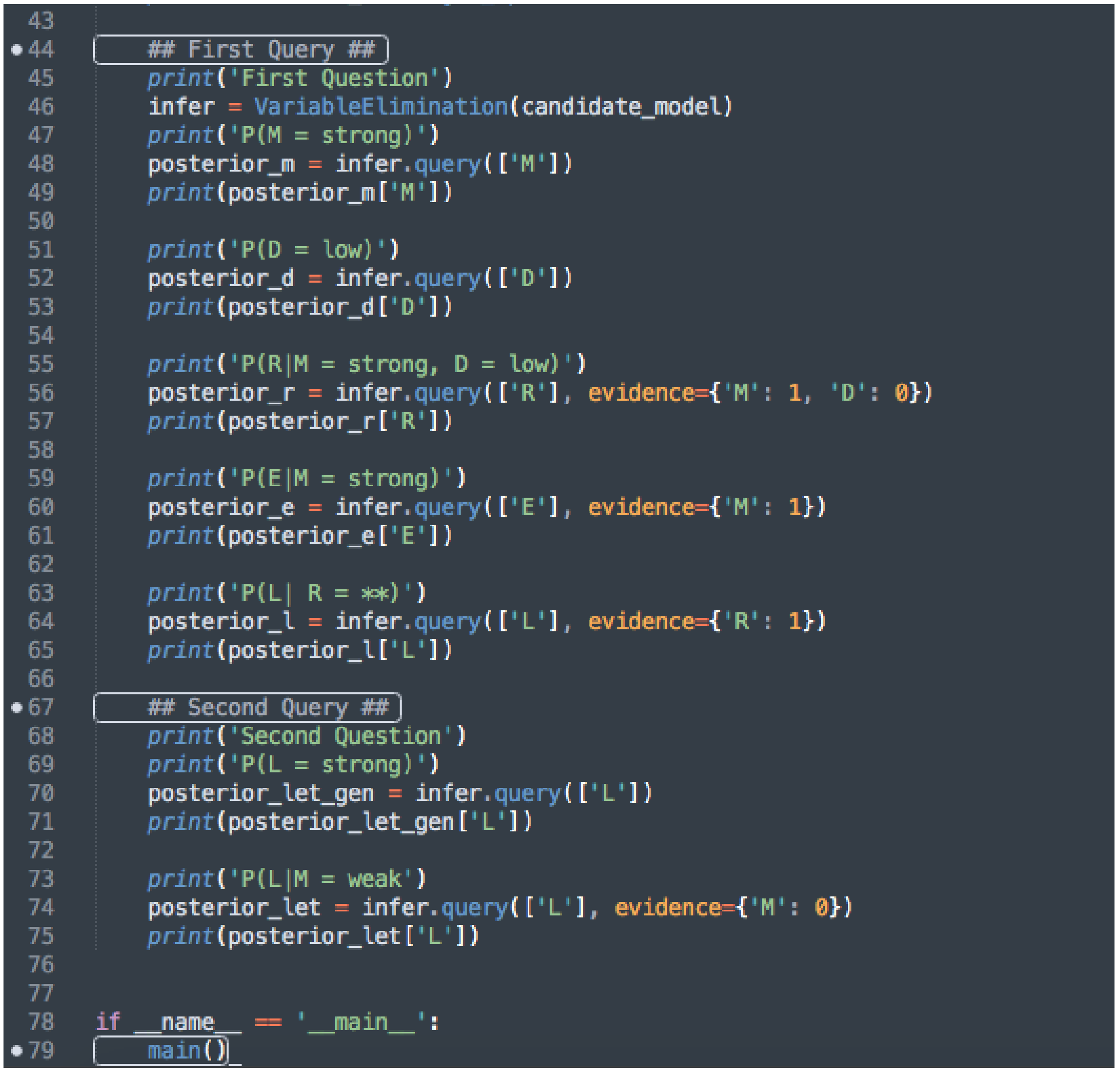
```
Andrews-Air:CSC 421 andrewyang$ python3 accuracy_a3.py
Train: [100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135
 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153
 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171
 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189
 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225
 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243
 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261
 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279
 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297
 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315
 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333
 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351
 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369
 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387
 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405
 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423
 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441
 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459
 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477
 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495
 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513
 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531
 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549
 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567
 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585
 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603
 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621
 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639
 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657
 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675
 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693
 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711
 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729
 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747
 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765
 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783
 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801
 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819
 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837
 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855
 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873
 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891
 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909
 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927
 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945
 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963
 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981
 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999] Test: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 96 97 98 99]
Traceback (most recent call last):
  File "accuracy_a3.py", line 32, in <module>
    main()
  File "accuracy_a3.py", line 23, in main
    X_train, X_test = X[train_index], X[test_index]
TypeError: only integer scalar arrays can be converted to a scalar index
Andrews-Air:CSC 421 andrewyang$ 
```

Part 3:

Code:

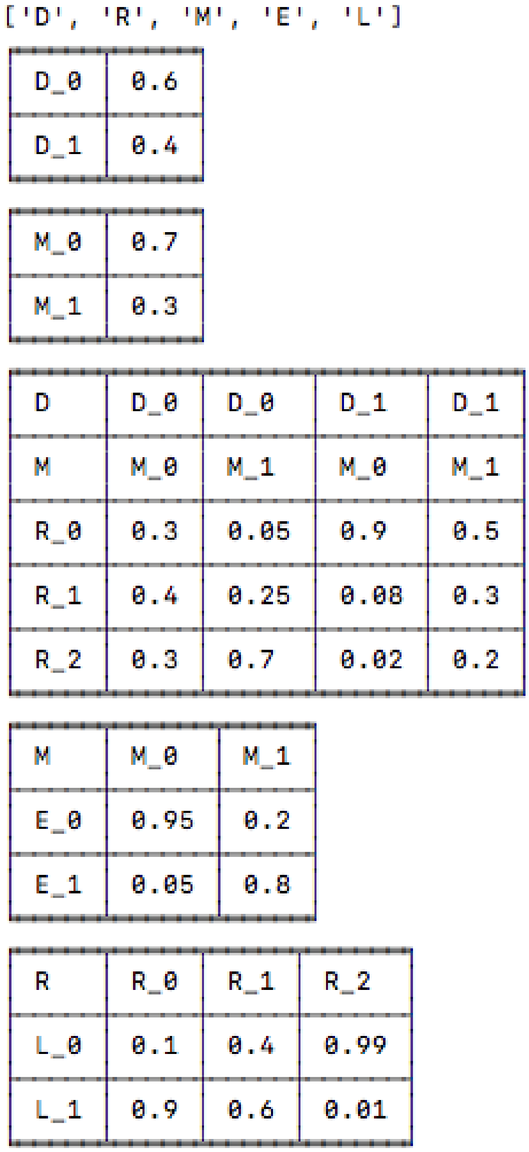


```python
from pgmpy.models import BayesianModel
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination

def main():
    # Bayesian Network Structure
    # Rating depends on Difficulty and Musicianship,
    # Exam depends on Musicianship
    # Letter depends on Rating
    candidate_model = BayesianModel([('D', 'R'), ('M', 'R'),
                                     ('M', 'E'), ('R', 'L')])

    # CPD's for each node
    difficulty_cpd = TabularCPD(variable='D', variable_card=2,
                                values=[[0.6, 0.4]])
    musicianship_cpd = TabularCPD(variable='M', variable_card=2,
                                  values=[[0.7, 0.3]])
    exam_cpd = TabularCPD(variable='E', variable_card=2,
                          values=[[0.95, 0.2], [0.05, 0.8]],
                          evidence=['M'], evidence_card=[2])
    rating_cpd = TabularCPD(variable='R', variable_card=3,
                            values=[[0.3, 0.05, 0.9, 0.5],
                                    [0.4, 0.25, 0.08, 0.3],
                                    [0.3, 0.7, 0.02, 0.2]],
                            evidence=['D', 'M'], evidence_card=[2, 2])
    letter_cpd = TabularCPD(variable='L', variable_card=2,
                            values=[[0.1, 0.4, 0.99], [0.9, 0.6, 0.01]],
                            evidence=['R'], evidence_card=[3])

    candidate_model.add_cpds(difficulty_cpd, musicianship_cpd, exam_cpd,
                             rating_cpd, letter_cpd)

    # Check if valid model
    candidate_model.check_model()

    # Print all the CPD's
    print(candidate_model.nodes())
    print(candidate_model.get_cpds('D'))
    print(candidate_model.get_cpds('M'))
    print(candidate_model.get_cpds('R'))
    print(candidate_model.get_cpds('E'))
    print(candidate_model.get_cpds('L'))
```



```python

    ## First Query ##
    print('First Question')
    infer = VariableElimination(candidate_model)
    print('P(M = strong)')
    posterior_m = infer.query(['M'])
    print(posterior_m['M'])

    print('P(D = low)')
    posterior_d = infer.query(['D'])
    print(posterior_d['D'])

    print('P(R|M = strong, D = low)')
    posterior_r = infer.query(['R'], evidence={'M': 1, 'D': 0})
    print(posterior_r['R'])

    print('P(E|M = strong)')
    posterior_e = infer.query(['E'], evidence={'M': 1})
    print(posterior_e['E'])

    print('P(L| R = **)')
    posterior_l = infer.query(['L'], evidence={'R': 1})
    print(posterior_l['L'])

    ## Second Query ##
    print('Second Question')
    print('P(L = strong)')
    posterior_let_gen = infer.query(['L'])
    print(posterior_let_gen['L'])

    print('P(L|M = weak')
    posterior_let = infer.query(['L'], evidence={'M': 0})
    print(posterior_let['L'])


if __name__ == '__main__':
    main()
```

Output:



```
['D', 'R', 'M', 'E', 'L']
```

D_0	0.6
D_1	0.4

M_0	0.7
M_1	0.3

D	D_0	D_0	D_1	D_1
M	M_0	M_1	M_0	M_1
R_0	0.3	0.05	0.9	0.5
R_1	0.4	0.25	0.08	0.3
R_2	0.3	0.7	0.02	0.2

M	M_0	M_1
E_0	0.95	0.2
E_1	0.05	0.8

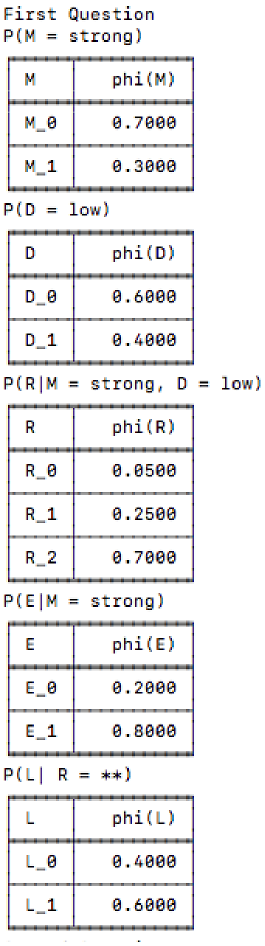
R	R_0	R_1	R_2
L_0	0.1	0.4	0.99
L_1	0.9	0.6	0.01

** The answers to 1, 2 and 3 will be posted below.

Question 1:

Using pgmpy we can map the Bayesian Network as detailed in the assignment. First the Bayesian Model is created in line 10. This line establishes the dependencies of each node. Then, the conditional probability tables are input from line 14 to 28. The results of the input are displayed above. Then the CPD's are added to our Bayesian Model.

Now with the model complete, the query can be computer by passing evidence to a Variable Elimination query. This occurs starting at line 44, where we see "First Query." Each query provides a new probability table based on the evidence provided. Solving the query is simply just finding each value and multiplying them together. The output of the queries is displayed below.

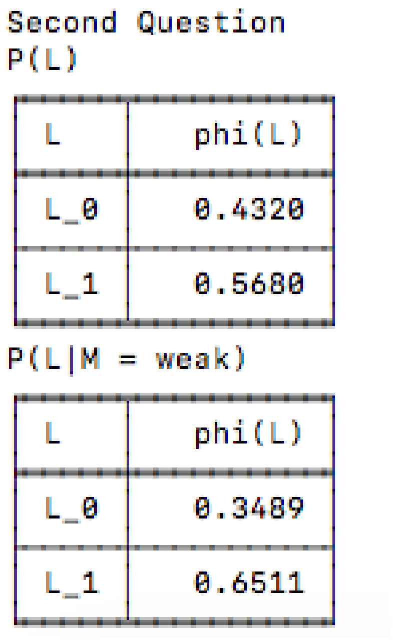


```
First Question
P(M = strong)
+-----+----------+
| M   |   phi(M) |
+=====+==========+
| M_0 |   0.7000 |
+-----+----------+
| M_1 |   0.3000 |
+-----+----------+
P(D = low)
+-----+----------+
| D   |   phi(D) |
+=====+==========+
| D_0 |   0.6000 |
+-----+----------+
| D_1 |   0.4000 |
+-----+----------+
P(R|M = strong, D = low)
+-----+----------+
| R   |   phi(R) |
+=====+==========+
| R_0 |   0.0500 |
+-----+----------+
| R_1 |   0.2500 |
+-----+----------+
| R_2 |   0.7000 |
+-----+----------+
P(E|M = strong)
+-----+----------+
| E   |   phi(E) |
+=====+==========+
| E_0 |   0.2000 |
+-----+----------+
| E_1 |   0.8000 |
+-----+----------+
P(L| R = **)
+-----+----------+
| L   |   phi(L) |
+=====+==========+
| L_0 |   0.4000 |
+-----+----------+
| L_1 |   0.6000 |
+-----+----------+
```

Question 2:

For this question, we can see that in general, George's chances of getting a strong recommendation letter is about 50.2%. This can be inferred by adding up all the probabilities of strong in our letter CPD and dividing by three. Using pgmpy, the queries are established starting line 67 and the section that begins with "Second Query."

These are the results from the queries.



```
Second Question
P(L)
+-----+----------+
| L   |   phi(L) |
+=====+==========+
| L_0 |   0.4320 |
+-----+----------+
| L_1 |   0.5680 |
+-----+----------+
P(L|M = weak)
+-----+----------+
| L   |   phi(L) |
+=====+==========+
| L_0 |   0.3489 |
+-----+----------+
| L_1 |   0.6511 |
+-----+----------+
```

As you can see, they aren't exact to the numbers that are listed in the assignment. The first query is close, however, the second one is off by almost double. I'm not exactly sure why this is.

Question 3:

These are solved using Variable Elimination and the results are displayed above in each question, along with the code snippets above at the beginning of this section.