

```

#!/usr/bin/env python3
import asyncio
import sys, os
import numpy as np
import chess.pgn
import chess.engine
import math
import itertools
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime
#Up There are the dependencies exluding re, sys, and os the rest are standard
#Here is where the pgn files are from https://theweekinchess.com/a-year-of-pgn-game-files

numFiles = len(sys.argv)
iterableFiles = range(1, numFiles)
baseName = [sys.argv[i][0:-4] for i in iterableFiles]
log2 = lambda a: math.log(a, 2)

def main():
    #This is the time which is the name of each file so they are unqiue each time
    runtime = datetime.now()
    dt_string = runtime.strftime("%d-%H-%M-%S")
    #the data we graph is in a dictionary
    data = {}
    #These go in the dictionary as a list of lists
    totalMoves = []
    totalShannonEntropies = []
    totalConditionalEntropies = []
    totalJointEntropy = []
    #open our data file which we write our statistics to
    dataFile = open(f"run-data-{dt_string}.out", 'w')
    #This iterates over all of the pgn files
    engine = chess.engine.SimpleEngine.popen_uci("/usr/games/stockfish")
    for i in iterableFiles:
        with open(sys.argv[i]) as pgn:
            #reads in the pgn and makes a chess board
            game = chess.pgn.read_game(pgn)
            board = game.board()
            dataFile.write(f"Game {i} and File Name {baseName[i-1]}\n")
            for moveNum, move in enumerate(game.mainline_moves()):
                if moveNum % 2 != 0:
                    #This only does the calculations if it is the white players move
                    moveTo = (str(move)[2::])
                    if len(moveTo) == 3:
                        moveTo = moveTo[:1]
                        piece = board.piece_at(chess.parse_square(moveTo))

                    weights = []
                    if ((numBetterMoves := len(list(board.legal_moves))) > 0):
                        info = engine.analyse(board, chess.engine.Limit(time=0.1, depth=10), multipv=numBetterMoves)
                        for j in info:
                            score = (j['score'])
                            num = str(score.black())
                            if num[0] == "#":
                                num = int(num[1:])
                            else:
                                num = int(num)
                            if (num > 0):
                                weights.append(num)
                        denom = sum(weights)
                        probabilities = [(val/denom) for val in weights]
                    else:
                        probabilities=[0]

                    # print("Weights ", weights)
                    # print("Probabilities", probabilities)
                    # print("Board Push")
                    board.push(move)
                    #round two

                    weights = []
                    if ((newNumBetterMoves := len(list(board.legal_moves))) > 0):
                        info = engine.analyse(board, chess.engine.Limit(time=0.1, depth=10), multipv=newNumBetterMoves)
                        for k in info:
                            score = (k['score'])
                            num = str(score.black())
                            if num[0] == "#":
                                num = int(num[1:])
                            else:
                                num = int(num)
                            if (num > 0):
                                weights.append(num)
                        denom = sum(weights)
                        newProbabilities = list([(val/denom) for val in weights])
                    else:
                        newProbabilities=[0]
                    # this calculate the probabilities of the new moves
                    # print("Weights ", weights)
                    # print("Probabilities", newProbabilities)
                    # print("Board Push")
                    blackMove = moveNum/2
                    entropy = shannonEntropy(probabilities)
                    conEntropy = conditionalEntropy(probabilities, newProbabilities)
                    jEntropy = jointEntropy(probabilities, newProbabilities)
                    dataFile.write(f"Player: Black Move Number: {blackMove} Move: {move} Piece: {piece} Number of Better Moves: {numBetterMoves} Entropy: {entropy} Conditional Entropy: {conEntropy} Joint I
                    totalMoves.append(blackMove)
                    totalShannonEntropies.append(shannonEntropy(probabilities))
                    totalConditionalEntropies.append(conEntropy)
                    totalJointEntropy.append(jEntropy)
                else:
                    board.push(move)
            #updates the data with the data filename:[stats, stats, stats]
            data.update({baseName[i-1]: [totalMoves, totalShannonEntropies, totalConditionalEntropies, totalJointEntropy]})
            #data.update({baseName[0] : [totalMoves, totalShannonEntropies]})

```

```

105         #data.update({baseName[0] : [totalMoves, totalShannonEntropies, totalConditionalEntropies]})
106         #empties the lists so we can fill and repeat with the next pgm file
107         totalMoves = []
108         totalShannonEntropies = []
109         totalConditionalEntropies = []
110         totalJointEntropy = []
111         print(i)
112     engine.quit()
113     dataFile.close()
114     plotShannonEntropy(data, dt_string)
115     plotBestFitShannonEntropy(data, dt_string)
116     plotConditionalEntropy(data, dt_string)
117     plotBestFitConditionalEntropy(data, dt_string)
118     plotJointEntropy(data, dt_string)
119     plotBestJointEntropy(data, dt_string)
120     exit()
121
122 def conditionalEntropy(probabilities, newProbabilities):
123     conditionalEntropy = -1 * sum(conditionalEntropy := [pX*pY * (log2(pX)/(pX*pY)) for pX in probabilities for pY in newProbabilities if pY != 0 and pX != 0])
124     return conditionalEntropy
125
126
127 def shannonEntropy(probabilities):
128     entropy = -1 * sum(entropies := [((prob)*log2(prob)) for prob in probabilities if prob != 0])
129     return entropy
130
131 def jointEntropy(probabilities, newProbabilities):
132     jointEntropy = -1 * sum((pX * pY * log2(pX*pY)) for pX in probabilities for pY in newProbabilities if pX !=0 and pY != 0)
133     return jointEntropy
134
135
136
137 def plotConditionalEntropy(data, dt_string):
138     plt.figure()
139     for i in list(data.keys()):
140         plt.plot(data[i][0], data[i][2])
141     plt.xlabel('Moves')
142     plt.ylabel('Conditional Entropy (bits)')
143     plt.title('Entropy over Time in Chess')
144     plt.savefig(f"figure-conditional-{dt_string}.png")
145     plt.show()
146
147 def plotBestFitConditionalEntropy(data, dt_string):
148     bestFitXs = []
149     bestFitYs = []
150     plt.figure()
151     for i in list(data.keys()):
152         plt.scatter(data[i][0], data[i][2])
153         bestFitXs.append(data[i][0])
154         bestFitYs.append(data[i][2])
155     bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
156     bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
157     model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
158     print(model)
159     nModel = np.polyder(model)
160     print(nModel)
161     plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
162     plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
163     plt.legend(handlelength=3)
164     plt.xlabel('Moves')
165     plt.ylabel('Conditional Entropy (bits)')
166     plt.title('Conditional Entropy During the 2021 World Chess Championships', fontsize = 20)
167     plt.savefig(f"figure-conditional-best-{dt_string}.png")
168     plt.show()
169
170 def plotBestFitShannonEntropy(data, dt_string):
171     bestFitXs = []
172     bestFitYs = []
173     plt.figure()
174     for i in list(data.keys()):
175         plt.scatter(data[i][0], data[i][1])
176         bestFitXs.append(data[i][0])
177         bestFitYs.append(data[i][1])
178     bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
179     bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
180     model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
181     print(model)
182     nModel = np.polyder(model)
183     print(nModel)
184     plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
185     plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
186     plt.legend(handlelength=3)
187     plt.xlabel('Moves')
188     plt.ylabel('Shannonian Entropy (bits)')
189     plt.title('Shannon Entropy During the 2021 World Chess Championships', fontsize = 20)
190     plt.savefig(f"figure-shannon-best-{dt_string}.png")
191     plt.show()
192
193 def plotShannonEntropy(data, dt_string):
194     plt.figure()
195     for i in list(data.keys()):
196         plt.plot(data[i][0], data[i][1])
197     plt.xlabel('Moves')
198
199     plt.ylabel('Shannonian Entropy')
200     plt.title('Entropy over Time in Chess')
201     plt.savefig(f"figure-shannon-{dt_string}.png")
202     plt.show()
203
204
205 def plotBestJointEntropy(data, dt_string):
206     bestFitXs = []
207     bestFitYs = []
208     plt.figure()
209     for i in list(data.keys()):
210         plt.scatter(data[i][0], data[i][3])
211         bestFitXs.append(data[i][0])
212         bestFitYs.append(data[i][3])

```

```

212     bestFitYs.append(data[i][3])
213 bestFitXs = np.array(list(itertools.chain("bestFitXs")))
214 bestFitYs = np.array(list(itertools.chain("bestFitYs")))
215 model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
216 print(model)
217 nModel = np.polyder(model)
218 print(nModel)
219 plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
220 plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
221 plt.legend(handlelength=3)
222 plt.xlabel('Moves')
223 plt.ylabel('Joint Entropy (bits)')
224 plt.title('Joint Entropy During the 2021 World Chess Championships', fontsize = 20)
225 plt.savefig(f"figure-joint-best-{dt_string}.png")
226 plt.show()
227
228 def plotJointEntropy(data, dt_string):
229     plt.figure()
230     for i in list(data.keys()):
231         plt.plot(data[i][0], data[i][3])
232     plt.xlabel('Moves')
233     plt.ylabel('Joint Entropy')
234     plt.title('Entropy over Time in Chess')
235     plt.savefig(f"figure-joint-{dt_string}.png")
236     plt.show()
237
238 if __name__ == "__main__":
239     main()

```

```

#!/usr/bin/env python3
import asyncio
import sys, os
1 import numpy as np
2 import chess.pgn
3 import chess.engine
4 import math
5 import itertools
6 import matplotlib.pyplot as plt
7 import pandas as pd
8 from datetime import datetime
9 #Up There are the dependencies exluding re, sys, and os the rest are standard
10 #Here is where the pgn files are from https://theweekinchess.com/a-year-of-pgn-game-files
11
12 numFiles = len(sys.argv)
13 iterableFiles = range(1, numFiles)
14 baseName = [sys.argv[i][0:-4] for i in iterableFiles]
15 log2 = lambda a: math.log(a, 2)
16
17
18 def main():
19     #This is the time which is the name of each file so they are unquie each time
20     runtime = datetime.now()
21     dt_string = runtime.strftime("%d-%H-%M-%S")
22     #the data we graph is in a dictionary
23     data = {}
24     #These go in the dictionary as a list of lists
25     totalMoves = []
26     totalShannonEntropies = []
27     totalConditionalEntropies = []
28     totalJointEntropy = []
29     #open our data file which we write our statistics to
30     dataFile = open(f"run-data-{dt_string}.out", 'w')
31     #This iterates over all of the pgn files
32     engine = chess.engine.SimpleEngine.popen_uci("/usr/games/stockfish")
33     for i in iterableFiles:
34         with open(sys.argv[i]) as pgn:
35             #reads in the pgn and makes a chess board
36             game = chess.pgn.read_game(pgn)
37             board = game.board()
38             dataFile.write(f"Game {i} and File Name {baseName[i-1]}\n")
39             for moveNum, move in enumerate(game.mainline_moves()):
40                 if moveNum % 2 != 0 and moveNum != 0:
41                     #This only does the calculations if it is the white players move
42                     moveTo = (str(move)[2::])
43                     if len(moveTo) == 3:
44                         moveTo = moveTo[:1]
45                         piece = board.piece_at(chess.parse_square(moveTo))
46
47                     # this calculate the probabilities of the new moves
48                     weights = []
49                     if ((numBetterMoves := len(list(board.legal_moves))) > 0):
50                         info = engine.analyse(board, chess.engine.Limit(time=0.1, depth=10), multipv=numBetterMoves)
51                         for j in info:
52                             score = (j['score'])
53                             num = str(score.black())
54                             if num[0] == "#":
55                                 num = int(num[1:])
56                             else:
57                                 num = int(num)
58                             if (num > 0):
59                                 weights.append(num)
60                         denom = sum(weights)
61                         probabilities = [(val/denom) for val in weights]
62                     else:
63
64                         probabilities=[0]
65                         # print("Weights ", weights)
66                         # print("Probabilities", probabilities)
67                         # print("Board Push")
68                         board.push(move)
69                         #round two
70
71                     weights = []
72                     if ((newNumBetterMoves := len(list(board.legal_moves))) > 0):
73                         info = engine.analyse(board, chess.engine.Limit(time=0.1, depth=10), multipv=newNumBetterMoves)
74                         for k in info:
75                             score = (k['score'])
76                             num = str(score.black())
77                             if num[0] == "#":
78                                 num = int(num[1:])
79                             else:
80                                 num = int(num)
81                             if (num > 0):
82                                 weights.append(num)
83                         denom = sum(weights)
84                         newProbabilities = list([(val/denom) for val in weights])
85                     else:
86                         newProbabilities=[0]
87                         # print("Weights ", weights)
88                         # print("Probabilities", newProbabilities)
89                         # print("Board Push")
90                         entropy = shannonEntropy(probabilities)
91                         conEntropy = conditionalEntropy(probabilities, newProbabilities)
92                         jEntropy = jointEntropy(probabilities, newProbabilities)
93                         dataFile.write(f"Player: Black Move Number: {moveNum} Move: {move} Piece: {piece} Number of Better Moves: {numBetterMoves} Entropy: {entropy} Conditional Entropy: {conEntropy} Joint E
94                         totalMoves.append(moveNum)
95                         totalShannonEntropies.append(shannonEntropy(probabilities))
96                         totalConditionalEntropies.append(conEntropy)
97                         totalJointEntropy.append(jEntropy)
98                     else:
99                         #This only does the calculations if it is the white players move
100                         moveTo = (str(move)[2::])
101                         if len(moveTo) == 3:
102                             moveTo = moveTo[:1]
103                         piece = board.piece_at(chess.parse_square(moveTo))
104

```

```

104
105     # this calculate the probabilities of the new moves
106     weights = []
107
108     if ((numBetterMoves := len(list(board.legal_moves))) > 0):
109         info = engine.analyse(board, chess.engine.Limit(time=0.1, depth=10), multipv=numBetterMoves)
110         for j in info:
111             score = (j['score'])
112             num = str(score.white())
113             if num[0] == "#":
114                 num = int(num[1:])
115             else:
116                 num = int(num)
117             if (num > 0):
118                 weights.append(num)
119         denom = sum(weights)
120         probabilities = [(val/denom) for val in weights]
121     else:
122         probabilities=[0]
123     # print("Weights ", weights)
124     # print("Probabilities", probabilities)
125     # print("Board Push")
126     board.push(move)
127     #round two
128
129     weights = []
130     if ((newNumBetterMoves := len(list(board.legal_moves))) > 0):
131         info = engine.analyse(board, chess.engine.Limit(time=0.1, depth=10), multipv=newNumBetterMoves)
132         for k in info:
133             score = (k['score'])
134             num = str(score.white())
135             if num[0] == "#":
136                 num = int(num[1:])
137             else:
138                 num = int(num)
139             if (num > 0):
140                 weights.append(num)
141         denom = sum(weights)
142         newProbabilities = list([(val/denom) for val in weights])
143     else:
144         newProbabilities=[0]
145     # print("Weights ", weights)
146     # print("Probabilities", newProbabilities)
147     # print("Board Push")
148     entropy = shannonEntropy(probabilities)
149     conEntropy = conditionalEntropy(probabilities, newProbabilities)
150     jEntropy = jointEntropy(probabilities, newProbabilities)
151     dataFile.write(f"Player: White Move Number: {moveNum} Move: {move} Piece: {piece} Number of Better Moves: {numBetterMoves} Entropy: {entropy} Conditional Entropy: {conEntropy} Joint Entropy: {jEntropy}\n")
152     totalMoves.append(moveNum)
153     totalShannonEntropies.append(shannonEntropy(probabilities))
154     totalConditionalEntropies.append(conEntropy)
155     totalJointEntropy.append(jEntropy)
156     #updates the data with the data filename:[stats, stats, stats]
157     data.update({baseName[i-1]: [totalMoves, totalShannonEntropies, totalConditionalEntropies, totalJointEntropy]})
158     #data.update({baseName[0]: [totalMoves, totalShannonEntropies]})
159     #data.update({baseName[0]: [totalMoves, totalShannonEntropies, totalConditionalEntropies]})
160     #empties the lists so we can fill and repeat with the next pgn file
161     totalMoves = []
162     totalShannonEntropies = []
163     totalConditionalEntropies = []
164     totalJointEntropy = []
165     print(f"File completion ({i})")
166     engine.quit()
167     dataFile.close()
168     plotShannonEntropy(data, dt_string)
169     plotBestFitShannonEntropy(data, dt_string)
170     plotConditionalEntropy(data, dt_string)
171     plotBestFitConditionalEntropy(data, dt_string)
172     plotJointEntropy(data, dt_string)
173     plotBestFitJointEntropy(data, dt_string)
174     exit()
175
176
177 def conditionalEntropy(probabilities, newProbabilities):
178     conditionalEntropy = -1 * sum(conditionalEntropy := [pX*pY * (log2(pX)/(pX*pY)) for pX in probabilities for pY in newProbabilities if pY != 0 and pX != 0])
179     return conditionalEntropy
180
181 def shannonEntropy(probabilities):
182     entropy = -1 * sum(entropies := [((prob)*log2(prob)) for prob in probabilities if prob != 0])
183     return entropy
184
185 def jointEntropy(probabilities, newProbabilities):
186     jointEntropy = -1 * sum((pX * pY * log2(pX*pY)) for pX in probabilities for pY in newProbabilities if pX !=0 and pY != 0)
187     return jointEntropy
188
189
190
191 def plotConditionalEntropy(data, dt_string):
192     plt.figure()
193     for i in list(data.keys()):
194         plt.plot(data[i][0], data[i][2])
195     plt.xlabel('Moves')
196     plt.ylabel('Conditional Entropy (bits)')
197
198     plt.title('Entropy over Time in Chess')
199     plt.savefig(f"figure-conditional-{dt_string}.png")
200     plt.show()
201
202 def plotBestFitConditionalEntropy(data, dt_string):
203     bestFitXs = []
204     bestFitYs = []
205     plt.figure()
206     for i in list(data.keys()):
207         plt.scatter(data[i][0], data[i][2])
208         bestFitXs.append(data[i][0])
209         bestFitYs.append(data[i][2])
210     bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
211     bestFitYs = np.array(list(itertools.chain(*bestFitYs)))

```

```

211 bestFitYs = np.array(list(itertools.chain('bestFitYs')))
212 model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
213 print(model)
214 nModel = np.polyder(model)
215 print(nModel)
216 plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
217 plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
218 plt.legend(handlelength=3)
219 plt.xlabel('Moves')
220 plt.ylabel('Conditional Entropy (bits)')
221 plt.title('Conditional Entropy During the 2021 World Chess Championships', fontsize = 20)
222 plt.savefig(f"figure-conditional-best-{dt_string}.png")
223 plt.show()
224
225 def plotBestFitShannonEntropy(data, dt_string):
226     bestFitXs = []
227     bestFitYs = []
228     plt.figure()
229     for i in list(data.keys()):
230         plt.scatter(data[i][0], data[i][1])
231         bestFitXs.append(data[i][0])
232         bestFitYs.append(data[i][1])
233     bestFitXs = np.array(list(itertools.chain("bestFitXs")))
234     bestFitYs = np.array(list(itertools.chain("bestFitYs")))
235     model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
236     print(model)
237     nModel = np.polyder(model)
238     print(nModel)
239     plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
240     plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
241     plt.legend(handlelength=3)
242     plt.xlabel('Moves')
243     plt.ylabel('Shannonian Entropy (bits)')
244     plt.title('Shannon Entropy During the 2021 World Chess Championships', fontsize = 20)
245     plt.savefig(f"figure-shannon-best-{dt_string}.png")
246     plt.show()
247
248 def plotShannonEntropy(data, dt_string):
249     plt.figure()
250     for i in list(data.keys()):
251         plt.plot(data[i][0], data[i][1])
252     plt.xlabel('Moves')
253     plt.ylabel('Shannonian Entropy')
254     plt.title('Entropy over Time in Chess')
255     plt.savefig(f"figure-shannon-{dt_string}.png")
256     plt.show()
257
258
259 def plotBestJointEntropy(data, dt_string):
260     bestFitXs = []
261     bestFitYs = []
262     plt.figure()
263     for i in list(data.keys()):
264
265         plt.scatter(data[i][0], data[i][3])
266         bestFitXs.append(data[i][0])
267         bestFitYs.append(data[i][3])
268     bestFitXs = np.array(list(itertools.chain("bestFitXs")))
269     bestFitYs = np.array(list(itertools.chain("bestFitYs")))
270     model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
271     print(model)
272     nModel = np.polyder(model)
273     print(nModel)
274     plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
275     plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
276     plt.legend(handlelength=3)
277     plt.xlabel('Moves')
278     plt.ylabel('Joint Entropy (bits)')
279     plt.title('Joint Entropy During the 2021 World Chess Championships', fontsize = 20)
280     plt.savefig(f"figure-joint-best-{dt_string}.png")
281     plt.show()
282
283 def plotJointEntropy(data, dt_string):
284     plt.figure()
285     for i in list(data.keys()):
286         plt.plot(data[i][0], data[i][3])
287     plt.xlabel('Moves')
288     plt.ylabel('Joint Entropy')
289     plt.title('Entropy over Time in Chess')
290     plt.savefig(f"figure-joint-{dt_string}.png")
291     plt.show()
292
293
294 if __name__ == "__main__":
295     main()

```

```

#!/usr/bin/env python3
import asyncio
import sys, os
import numpy as np
import chess.pgn
import chess.engine
import math
import itertools
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime
#Up There are the dependencies excluding re, sys, and os the rest are standard
#Here is where the pgn files are from https://theweekinchess.com/a-year-of-pgn-game-files

numFiles = len(sys.argv)
iterableFiles = range(1, numFiles)
baseName = [sys.argv[i][0:-4] for i in iterableFiles]
log2 = lambda a: math.log(a, 2)

def main():
    #This is the time which is the name of each file so they are unique each time
    runtime = datetime.now()
    dt_string = runtime.strftime("%d-%H-%M-%S")
    #the data we graph is in a dictionary
    data = {}
    #These go in the dictionary as a list of lists
    totalMoves = []
    totalShannonEntropies = []
    totalConditionalEntropies = []
    totalJointEntropy = []
    #open our data file which we write our statistics to
    dataFile = open("run-data-{}.out".format(dt_string), 'w')
    #This iterates over all of the pgn files
    engine = chess.engine.SimpleEngine.popen_uci("/usr/games/stockfish")
    for i in iterableFiles:
        with open(sys.argv[i]) as pgn:
            #reads in the pgn and makes a chess board
            game = chess.pgn.read_game(pgn)
            board = game.board()
            dataFile.write("Game {} and File Name {}".format(i, baseName[i-1])+"\n")
            for moveNum, move in enumerate(game.mainline_moves()):
                if moveNum % 2 == 0:
                    #This only does the calculations if it is the white players move
                    moveTo = (str(move)[2:])
                    if len(moveTo) == 3:
                        moveTo = moveTo[:1]
                        piece = board.piece_at(chess.parse_square(moveTo))

                    # this calculate the probabilities of the new moves
                    weights = []
                    if ((numBetterMoves := len(list(board.legal_moves))) > 0):
                        info = engine.analyse(board, chess.engine.Limit(time=0.1, depth=10), multipv=numBetterMoves)
                        for j in info:
                            score = (j['score'])
                            num = str(score.white())
                            if num[0] == "#":
                                num = int(num[1:])
                            else:
                                num = int(num)
                            if (num > 0):
                                weights.append(num)
                        denom = sum(weights)
                        probabilities = [(val/denom) for val in weights]
                    else:
                        probabilities=[0]
                    # print("Weights ", weights)
                    # print("Probabilities", probabilities)
                    # print("Board Push")
                    board.push(move)
                    #round two

                    weights = []
                    if ((newNumBetterMoves := len(list(board.legal_moves))) > 0):
                        info = engine.analyse(board, chess.engine.Limit(time=0.1, depth=10), multipv=newNumBetterMoves)
                        for k in info:
                            score = (k['score'])
                            num = str(score.white())
                            if num[0] == "#":
                                num = int(num[1:])
                            else:
                                num = int(num)
                            if (num > 0):
                                weights.append(num)
                        denom = sum(weights)
                        newProbabilities = list([(val/denom) for val in weights])
                    else:
                        newProbabilities=[0]
                    # print("Weights ", weights)
                    # print("Probabilities", newProbabilities)
                    # print("Board Push")
                    whiteMove = moveNum/2
                    entropy = shannonEntropy(probabilities)
                    conEntropy = conditionalEntropy(probabilities, newProbabilities)
                    jEntropy = jointEntropy(probabilities, newProbabilities)
                    dataFile.write("Player: White Move Number: {} Move: {} Piece: {} Number of Better Moves: {} Entropy: {} Conditional Entropy: {} Joint I".format(whiteMove, move, piece, numBetterMoves, entropy, conEntropy, jEntropy))
                    totalMoves.append(whiteMove)
                    totalShannonEntropies.append(shannonEntropy(probabilities))
                    totalConditionalEntropies.append(conEntropy)
                    totalJointEntropy.append(jEntropy)
                else:
                    board.push(move)
            #updates the data with the data filename:[stats, stats, stats]
            data.update({baseName[i-1]: [totalMoves, totalShannonEntropies, totalConditionalEntropies, totalJointEntropy]})
            #data.update({baseName[0] : [totalMoves, totalShannonEntropies]})

```

```

105     #data.update({baseName[j]: [totalMoves, totalShannonEntropies, totalConditionalEntropies]})
106     #empties the lists so we can fill and repeat with the next pgm file
107     totalMoves = []
108     totalShannonEntropies = []
109     totalConditionalEntropies = []
110     totalJointEntropy = []
111     print(i)
112 engine.quit()
113 dataFile.close()
114 plotShannonEntropy(data, dt_string)
115 plotBestFitShannonEntropy(data, dt_string)
116 plotConditionalEntropy(data, dt_string)
117 plotBestFitConditionalEntropy(data, dt_string)
118 plotJointEntropy(data, dt_string)
119 plotBestJointEntropy(data, dt_string)
120 exit()
121
122
123 def conditionalEntropy(probabilities, newProbabilities):
124     conditionalEntropy = -1 * sum(conditionalEntropy := [pX*pY * (log2(pX)/(pX*pY)) for pX in probabilities for pY in newProbabilities if pY != 0 and pX != 0])
125     return conditionalEntropy
126
127
128 def shannonEntropy(probabilities):
129     entropy = -1 * sum(entropies := [(prob)*log2(prob)) for prob in probabilities if prob != 0])
130     return entropy
131
132
133 def jointEntropy(probabilities, newProbabilities):
134     jointEntropy = -1 * sum((pX * pY * log2(pX*pY)) for pX in probabilities for pY in newProbabilities if pX !=0 and pY != 0)
135     return jointEntropy
136
137
138 def plotConditionalEntropy(data, dt_string):
139     plt.figure()
140     for i in list(data.keys()):
141         plt.plot(data[i][0], data[i][2])
142     plt.xlabel('Moves')
143     plt.ylabel('Conditional Entropy (bits)')
144     plt.title('Entropy over Time in Chess')
145     plt.savefig(f"figure-conditional-{dt_string}.png")
146     plt.show()
147
148 def plotBestFitConditionalEntropy(data, dt_string):
149     bestFitXs = []
150     bestFitYs = []
151     plt.figure()
152     for i in list(data.keys()):
153         plt.scatter(data[i][0], data[i][2])
154         bestFitXs.append(data[i][0])
155         bestFitYs.append(data[i][2])
156     bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
157     bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
158     model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
159     print(model)
160     nModel = np.polyder(model)
161     print(nModel)
162     plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
163     plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
164     plt.legend(handlelength=3)
165     plt.xlabel('Moves')
166     plt.ylabel('Conditional Entropy (bits)')
167     plt.title('Conditional Entropy During the 2021 World Chess Championships', fontsize = 20)
168     plt.savefig(f"figure-conditional-best-{dt_string}.png")
169     plt.show()
170
171 def plotBestFitShannonEntropy(data, dt_string):
172     bestFitXs = []
173     bestFitYs = []
174     plt.figure()
175     for i in list(data.keys()):
176         plt.scatter(data[i][0], data[i][1])
177         bestFitXs.append(data[i][0])
178         bestFitYs.append(data[i][1])
179     bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
180     bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
181     model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
182     print(model)
183     nModel = np.polyder(model)
184     print(nModel)
185     plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
186     plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
187     plt.legend(handlelength=3)
188     plt.xlabel('Moves')
189     plt.ylabel('Shannonian Entropy (bits)')
190     plt.title('Shannon Entropy During the 2021 World Chess Championships', fontsize = 20)
191     plt.savefig(f"figure-shannon-best-{dt_string}.png")
192     plt.show()
193
194 def plotShannonEntropy(data, dt_string):
195     plt.figure()
196     for i in list(data.keys()):
197         plt.plot(data[i][0], data[i][1])
198
199     plt.xlabel('Moves')
200     plt.ylabel('Shannonian Entropy')
201     plt.title('Entropy over Time in Chess')
202     plt.savefig(f"figure-shannon-{dt_string}.png")
203     plt.show()
204
205 def plotBestJointEntropy(data, dt_string):
206     bestFitXs = []
207     bestFitYs = []
208     plt.figure()
209     for i in list(data.keys()):
210         plt.scatter(data[i][0], data[i][3])
211         bestFitXs.append(data[i][0])
212         bestFitYs.append(data[i][3])
213     bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
214     bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
215     model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
216     print(model)
217     nModel = np.polyder(model)
218     print(nModel)
219     plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
220     plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
221     plt.legend(handlelength=3)
222     plt.xlabel('Moves')
223     plt.ylabel('Joint Entropy (bits)')
224     plt.title('Joint Entropy During the 2021 World Chess Championships', fontsize = 20)
225     plt.savefig(f"figure-joint-best-{dt_string}.png")
226     plt.show()

```

```

212     bestFitXs.append(data[i][0])
213     bestFitYs.append(data[i][3])
214     bestFitXs = np.array(list(itertools.chain("bestFitXs")))
215     bestFitYs = np.array(list(itertools.chain("bestFitYs")))
216     model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
217     print(model)
218     nModel = np.polyder(model)
219     print(nModel)
220     plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
221     plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
222     plt.legend(handlelength=3)
223     plt.xlabel('Moves')
224     plt.ylabel('Joint Entropy (bits)')
225     plt.title('Joint Entropy During the 2021 World Chess Championships', fontsize = 20)
226     plt.savefig(f"figure-joint-best-{dt_string}.png")
227     plt.show()
228
229 def plotJointEntropy(data, dt_string):
230     plt.figure()
231     for i in list(data.keys()):
232         plt.plot(data[i][0], data[i][3])
233     plt.xlabel('Moves')
234     plt.ylabel('Joint Entropy')
235     plt.title('Entropy over Time in Chess')
236     plt.savefig(f"figure-joint-{dt_string}.png")
237     plt.show()
238
239
240 if __name__ == "__main__":
241     main()

```

```

#!/usr/bin/env python3
import sys, os
import numpy as np
import chess.pgn
import math
import itertools
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime

#Up There are the dependencies excluding re, sys, and os the rest are standard
#Here is where the pgn files are from https://theweekinchess.com/a-year-of-pgn-game-files

numFiles = len(sys.argv)
iterableFiles = range(1, numFiles)
baseName = [sys.argv[i][0:-4] for i in iterableFiles]
log2 = lambda a: math.log(a, 2)

def main():
    #This is the time which is the name of each file so they are unique each time
    runtime = datetime.now()
    dt_string = runtime.strftime("%d-%H-%M-%S")
    #the data we graph is in a dictionary
    data = {}
    #These go in the dictionary as a list of lists
    totalMoves = []
    totalShannonEntropies = []
    totalConditionalEntropies = []
    totalJointEntropy = []
    #open our data file which we write our statistics to
    dataFile = open(f"run-data-{dt_string}.out", 'w')
    #This iterates over all of the pgn files
    for i in iterableFiles:
        with open(sys.argv[i]) as pgn:
            #reads in the pgn and makes a chess board
            game = chess.pgn.read_game(pgn)
            board = game.board()
            dataFile.write(f"Game {i} and File Name {baseName[i-1]}\n")
            for moveNum, move in enumerate(game.mainline_moves()):
                if moveNum % 2 != 0:
                    #This identifies the piece on the square we move to
                    moveTo = (str(move)[2::])
                    if len(moveTo) == 3:
                        moveTo = moveTo[:1]
                        piece = board.piece_at(chess.parse_square(moveTo))
                        #calculates the number of moves and the list of moves
                        legalMoves = list(board.legal_moves)
                        numLegalMoves = len(list(board.legal_moves))
                        #writes the data alternating white and then black
                        #pushes the board state to the next one
                        board.push(move)
                        #calculates the new set of moves for conditional entropy
                        newNumMovesCon = len(list(board.legal_moves))
                        dataFile.write(f"Player: Black Move Number: {moveNum // 2} Move: {move} Piece: {piece} Number of Legal Moves: {numLegalMoves} Entropy: {shannonEntropy(numLegalMoves)} Conditional")
                        #append the moves and entropies to our data
                        totalMoves.append(moveNum//2)
                        totalShannonEntropies.append(shannonEntropy(numLegalMoves))
                        totalConditionalEntropies.append(conditionalEntropy(numLegalMoves, newNumMovesCon))
                        totalJointEntropy.append(jointEntropy(numLegalMoves, newNumMovesCon))
                        #updates the data with the data filename:[stats, stats, stats]
                    else:
                        board.push(move)
            data.update({baseName[i-1]: [totalMoves, totalShannonEntropies, totalConditionalEntropies, totalJointEntropy]})
            #empties the lists so we can fill and repeat with the next pgn file
            totalMoves = []
            totalShannonEntropies = []
            totalConditionalEntropies = []
            totalJointEntropy = []
    #close the file
    dataFile.close()
    #plots different junk
    plotShannonEntropy(data, dt_string)
    plotBestFitShannonEntropy(data, dt_string)
    plotConditionalEntropy(data, dt_string)
    plotBestFitConditionalEntropy(data, dt_string)
    plotJointEntropy(data, dt_string)
    plotBestJointEntropy(data, dt_string)

#!/usr/bin/env python3
import sys, os
import numpy as np
import chess.pgn
import math
import itertools
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime

#Up There are the dependencies excluding re, sys, and os the rest are standard
#Here is where the pgn files are from https://theweekinchess.com/a-year-of-pgn-game-files

numFiles = len(sys.argv)
iterableFiles = range(1, numFiles)
baseName = [sys.argv[i][0:-4] for i in iterableFiles]
log2 = lambda a: math.log(a, 2)

def main():
    #This is the time which is the name of each file so they are unique each time
    runtime = datetime.now()
    dt_string = runtime.strftime("%d-%H-%M-%S")
    #the data we graph is in a dictionary
    data = {}
    #These go in the dictionary as a list of lists
    totalMoves = []
    totalShannonEntropies = []
    totalConditionalEntropies = []
    totalJointEntropy = []

```

```

106 #open our data file which we write our statistics to
107 dataFile = open(f"run-data-{dt_string}.out", 'w')
108 #This iterates over all of the pgn files
109 for i in iterableFiles:
110     with open(sys.argv[i]) as pgn:
111         #reads in the pgn and makes a chess board
112         game = chess.pgn.read_game(pgn)
113         board = game.board()
114         dataFile.write(f"Game {i} and File Name {baseName[i-1]}\n")
115         for moveNum, move in enumerate(game.mainline_moves()):
116             if moveNum % 2 != 0:
117                 #This identifies the piece on the square we move to
118                 moveTo = (str(move)[2::])
119                 if len(moveTo) == 3:
120                     moveTo = moveTo[:-1]
121                     piece = board.piece_at(chess.parse_square(moveTo))
122                     #calculates the number of moves and the list of moves
123                     legalMoves = list(board.legal_moves)
124                     numLegalMoves = len(list(board.legal_moves))
125                     #writes the data alternating white and then black
126                     #pushes the board state to the next one
127                     board.push(move)
128                     #calculates the new set of moves for conditional entropy
129                     newNumMovesCon = len(list(board.legal_moves))
130                     dataFile.write(f"Player: Black Move Number: {moveNum // 2} Move: {move} Piece: {piece} Number of Legal Moves: {numLegalMoves} Entropy: {shannonEntropy(numLegalMoves)} Conditional
131                     #append the moves and entropies to our data
132
133                     totalMoves.append(moveNum//2)
134                     totalShannonEntropies.append(shannonEntropy(numLegalMoves))
135                     totalConditionalEntropies.append(conditionalEntropy(numLegalMoves, newNumMovesCon))
136                     totalJointEntropy.append(jointEntropy(numLegalMoves, newNumMovesCon))
137                     #updates the data with the data filename:[stats, stats, stats]
138                 else:
139                     board.push(move)
140                     data.update({baseName[i-1]: [totalMoves, totalShannonEntropies, totalConditionalEntropies, totalJointEntropy]})
141                     #empties the lists so we can fill and repeat with the next pgn file
142                     totalMoves = []
143                     totalShannonEntropies = []
144                     totalConditionalEntropies = []
145                     totalJointEntropy = []
146 #close the file
147 dataFile.close()
148 #plots different junk
149 plotShannonEntropy(data, dt_string)
150 plotBestFitShannonEntropy(data, dt_string)
151 plotConditionalEntropy(data, dt_string)
152 plotBestFitConditionalEntropy(data, dt_string)
153 plotJointEntropy(data, dt_string)
154 plotBestJointEntropy(data, dt_string)
155
156
157 def plotBestJointEntropy(data, dt_string):
158     bestFitXs = []
159     bestFitYs = []
160     plt.figure()
161     for i in list(data.keys()):
162         plt.scatter(data[i][0], data[i][3])
163         bestFitXs.append(data[i][0])
164         bestFitYs.append(data[i][3])
165     bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
166     bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
167     model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
168     print(model)
169     nModel = np.polyder(model)
170     print(nModel)
171     plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
172     plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
173     plt.legend(handlelength=3)
174     plt.xlabel('Moves')
175     plt.ylabel('Joint Entropy (bits)')
176     plt.title('Joint Entropy During the 2021 World Chess Championships', fontsize = 20)
177     plt.savefig(f"figure-joint-best-{dt_string}.png")
178     plt.show()
179
180 def plotJointEntropy(data, dt_string):
181     plt.figure()
182     for i in list(data.keys()):
183         plt.plot(data[i][0], data[i][3])
184     plt.xlabel('Moves')
185     plt.ylabel('Joint Entropy')
186     plt.title('Entropy over Time in Chess')
187     plt.savefig(f"figure-joint-{dt_string}.png")
188     plt.show()
189
190 def conditionalEntropy(probabilities, newProbabilities):
191     conditionalEntropy = -1 * sum(conditionalEntropy := [(1/probabilities)*(1/newProbabilities) * log2(((1/probabilities)*(1/newProbabilities)))/((1/probabilities)) for pX in range(1, probabilities+1) for pY in ra
192     return conditionalEntropy
193
194 def shannonEntropy(probabilities):
195     entropy = -1 * sum(entropies := [((1/probabilities)*log2(1/probabilities)) for prob in range(1, probabilities+1)])
196     return entropy
197
198
199 def jointEntropy(probabilities, newProbabilities):
200     jointEntropy = -1 * sum((((1/probabilities) * (1/newProbabilities)) * log2((1/probabilities)*(1/newProbabilities)) for pX in range(1, probabilities+1) for pY in range(1, newProbabilities+1)))
201     return jointEntropy
202
203
204 def plotConditionalEntropy(data, dt_string):
205     plt.figure()
206     for i in list(data.keys()):
207         plt.plot(data[i][0], data[i][2])
208     plt.xlabel('Moves')
209     plt.ylabel('Conditional Entropy (bits)')
210     plt.title('Entropy over Time in Chess')
211     plt.savefig(f"figure-conditional-{dt_string}.png")
212

```

```
212 plt.show()
213
214
215 def plotBestFitConditionalEntropy(data, dt_string):
216     bestFitXs = []
217     bestFitYs = []
218     plt.figure()
219     for i in list(data.keys()):
220         plt.scatter(data[i][0], data[i][2])
221         bestFitXs.append(data[i][0])
222         bestFitYs.append(data[i][2])
223     bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
224     bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
225     model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
226     print(model)
227     nModel = np.polyder(model)
228     print(nModel)
229     plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
230     plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
231     plt.legend(handlelength=3)
232     plt.xlabel('Moves')
233     plt.ylabel('Conditional Entropy (bits)')
234     plt.title('Conditional Entropy During the 2021 World Chess Championships', fontsize = 20)
235     plt.savefig(f"figure-conditional-best-{dt_string}.png")
236     plt.show()
237
238 def plotBestFitShannonEntropy(data, dt_string):
239     bestFitXs = []
240     bestFitYs = []
241     plt.figure()
242     for i in list(data.keys()):
243         plt.scatter(data[i][0], data[i][1])
244         bestFitXs.append(data[i][0])
245         bestFitYs.append(data[i][1])
246     bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
247     bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
248     model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
249     print(model)
250     nModel = np.polyder(model)
251     print(nModel)
252     plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
253     plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
254     plt.legend(handlelength=3)
255     plt.xlabel('Moves')
256     plt.ylabel('Shannonian Entropy (bits)')
257     plt.title('Shannon Entropy During the 2021 World Chess Championships', fontsize = 20)
258     plt.savefig(f"figure-shannon-best-{dt_string}.png")
259     plt.show()
260
261 def plotShannonEntropy(data, dt_string):
262     plt.figure()
263     for i in list(data.keys()):
264         plt.plot(data[i][0], data[i][1])
265     plt.xlabel('Moves')
266
267     plt.ylabel('Shannonian Entropy')
268     plt.title('Entropy over Time in Chess')
269     plt.savefig(f"figure-shannon-{dt_string}.png")
270     plt.show()
271
272 if __name__ == "__main__":
273     main()
```

```

#!/usr/bin/env python3
1 import sys, os
2 import numpy as np
3 import chess.pgn
4 import math
5 import itertools
6 import matplotlib.pyplot as plt
7 import pandas as pd
8 from datetime import datetime
9 #Up There are the dependencies exluding re, sys, and os the rest are standard
10 #Here is where the pgn files are from https://theweekinchess.com/a-year-of-pgn-game-files
11
12 numFiles = len(sys.argv)
13 iterableFiles = range(1, numFiles)
14 baseName = [sys.argv[i][0:-4] for i in iterableFiles]
15 log2 = lambda a: math.log(a, 2)
16
17 def main():
18     #This is the time which is the name of each file so they are unqiue each time
19     runtime = datetime.now()
20     dt_string = runtime.strftime("%d-%H-%M-%S")
21     #the data we graph is in a dictionary
22     data = {}
23     #These go in the dictionary as a list of lists
24     totalMoves = []
25     totalShannonEntropies = []
26     totalConditionalEntropies = []
27     totalJointEntropy = []
28     #open our data file which we write our statistics to
29     dataFile = open(f"run-data-{dt_string}.out", 'w')
30     #This iterates over all of the pgn files
31     for i in iterableFiles:
32         with open(sys.argv[i]) as pgn:
33             #reads in the pgn and makes a chess board
34             game = chess.pgn.read_game(pgn)
35             board = game.board()
36             dataFile.write(f"Game {i} and File Name {baseName[i-1]}\n")
37             for moveNum, move in enumerate(game.mainline_moves()):
38                 #This identifies the piece on the square we move t
39                 moveTo = (str(move)[2::])
40                 if len(moveTo) == 3:
41                     moveTo = moveTo[:-1]
42                 piece = board.piece_at(chess.parse_square(moveTo))
43                 #calculates the number of moves and the list of moves
44                 legalMoves = list(board.legal_moves)
45                 numLegalMoves = len(list(board.legal_moves))
46                 #writes the data alternating white and then black
47                 #pushes the board state to the next one
48                 board.push(move)
49                 #calculates the new set of moves for conditional entropy
50                 newNumMovesCon = len(list(board.legal_moves))
51                 if moveNum % 2 == 0:
52                     dataFile.write(f"Player: White Move Number: {moveNum} Move: {move} Piece: {piece} Number of Legal Moves: {numLegalMoves} Entropy: {shannonEntropy(numLegalMoves)} Conditional Entropy: {conditionalEntropy(numLegalMoves, newNumMovesCon)} Joint Entropy: {jointEntropy(numLegalMoves, newNumMovesCon)}\n")
53                 else:
54                     dataFile.write(f"Player: Black Move Number: {moveNum} Move: {move} Piece: {piece} Number of Legal Moves: {numLegalMoves} Entropy: {shannonEntropy(numLegalMoves)} Conditional Entropy: {conditionalEntropy(numLegalMoves, newNumMovesCon)} Joint Entropy: {jointEntropy(numLegalMoves, newNumMovesCon)}\n")
55                 #append the moves and entropies to our data
56                 totalMoves.append(moveNum)
57                 totalShannonEntropies.append(shannonEntropy(numLegalMoves))
58                 totalConditionalEntropies.append(conditionalEntropy(numLegalMoves, newNumMovesCon))
59                 totalJointEntropy.append(jointEntropy(numLegalMoves, newNumMovesCon))
60             #updates the data with the data filename:[stats, stats, stats]
61             data.update({baseName[i-1]: [totalMoves, totalShannonEntropies, totalConditionalEntropies, totalJointEntropy]})
62             #empties the lists so we can fill and repeat with the next pgn file
63             totalMoves = []
64             totalShannonEntropies = []
65             totalConditionalEntropies = []
66             totalJointEntropy = []
67             print(f"File {i} completion")
68     #close the file
69     dataFile.close()
70     #plots different junk
71     plotShannonEntropy(data, dt_string)
72     plotBestFitShannonEntropy(data, dt_string)
73     plotConditionalEntropy(data, dt_string)
74     plotBestFitConditionalEntropy(data, dt_string)
75     plotJointEntropy(data, dt_string)
76     plotBestJointEntropy(data, dt_string)
77
78
79
80 def plotBestJointEntropy(data, dt_string):
81     bestFitXs = []
82     bestFitYs = []
83     plt.figure()
84     for i in list(data.keys()):
85         plt.scatter(data[i][0], data[i][3])
86         bestFitXs.append(data[i][0])
87         bestFitYs.append(data[i][3])
88     bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
89     bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
90     model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
91     print(model)
92     nModel = np.polyder(model)
93     print(nModel)
94     plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
95     plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
96     plt.legend(handlelength=3)
97     plt.xlabel('Moves')
98     plt.ylabel('Joint Entropy (bits)')
99     plt.title('Joint Entropy During the 2021 World Chess Championships', fontsize = 20)
100     plt.savefig(f"figure-joint-best-{dt_string}.png")
101     plt.show()
102
103 def plotJointEntropy(data, dt_string):
104     plt.figure()
105     for i in list(data.keys()):
106         plt.scatter(data[i][0], data[i][4])
107         bestFitXs.append(data[i][0])
108         bestFitYs.append(data[i][4])
109     bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
110     bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
111     model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
112     print(model)
113     nModel = np.polyder(model)
114     print(nModel)
115     plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
116     plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
117     plt.legend(handlelength=3)
118     plt.xlabel('Moves')
119     plt.ylabel('Joint Entropy (bits)')
120     plt.title('Joint Entropy During the 2021 World Chess Championships', fontsize = 20)
121     plt.savefig(f"figure-joint-best-{dt_string}.png")
122     plt.show()

```

```
106     plt.plot(data[i][0], data[i][3])
107     plt.xlabel('Moves')
108     plt.ylabel('Joint Entropy')
109     plt.title('Entropy over Time in Chess')
110     plt.savefig(f"figure-joint-{dt_string}.png")
111     plt.show()
112
113 def conditionalEntropy(probabilities, newProbabilities):
114     conditionalEntropy = -1 * sum(conditionalEntropy := [(1/probabilities)*(1/newProbabilities) * log2(((1/probabilities)*(1/newProbabilities)))/((1/probabilities)) for pX in range(1, probabilities+1) for pY in range(1, newProbabilities+1)])
115     return conditionalEntropy
116
117 def shannonEntropy(probabilities):
118     entropy = -1 * sum(entropies := [((1/probabilities)*log2(1/probabilities)) for prob in range(1, probabilities+1)])
119     return entropy
120
121 def jointEntropy(probabilities, newProbabilities):
122     jointEntropy = -1 * sum(((1/probabilities) * (1/newProbabilities)) * log2((1/probabilities)*(1/newProbabilities)) for pX in range(1, probabilities+1) for pY in range(1, newProbabilities+1))
123     return jointEntropy
124
125
126
127 def plotConditionalEntropy(data, dt_string):
128     plt.figure()
129     for i in list(data.keys()):
130         plt.plot(data[i][0], data[i][2])
131     plt.xlabel('Moves')
132     plt.ylabel('Conditional Entropy (bits)')
133     plt.title('Entropy over Time in Chess')
134     plt.savefig(f"figure-conditional-{dt_string}.png")
135     plt.show()
136
137 def plotBestFitConditionalEntropy(data, dt_string):
138     bestFitXs = []
139     bestFitYs = []
140     plt.figure()
141     for i in list(data.keys()):
142         plt.scatter(data[i][0], data[i][2])
143         bestFitXs.append(data[i][0])
144         bestFitYs.append(data[i][2])
145     bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
146     bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
147     model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
148     print(model)
149     nModel = np.polyder(model)
150     print(nModel)
151     plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
152     plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
153     plt.legend(handlelength=3)
154     plt.xlabel('Moves')
155     plt.ylabel('Conditional Entropy (bits)')
156     plt.title('Conditional Entropy During the 2021 World Chess Championships', fontsize = 20)
157     plt.savefig(f"figure-conditional-best-{dt_string}.png")
158     plt.show()
159
160 def plotBestFitShannonEntropy(data, dt_string):
161     bestFitXs = []
162     bestFitYs = []
163     plt.figure()
164     for i in list(data.keys()):
165         plt.scatter(data[i][0], data[i][1])
166         bestFitXs.append(data[i][0])
167         bestFitYs.append(data[i][1])
168     bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
169     bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
170     model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
171     print(model)
172     nModel = np.polyder(model)
173     print(nModel)
174     plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
175     plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
176     plt.legend(handlelength=3)
177     plt.xlabel('Moves')
178     plt.ylabel('Shannonian Entropy (bits)')
179     plt.title('Shannon Entropy During the 2021 World Chess Championships', fontsize = 20)
180     plt.savefig(f"figure-shannon-best-{dt_string}.png")
181     plt.show()
182
183 def plotShannonEntropy(data, dt_string):
184     plt.figure()
185     for i in list(data.keys()):
186         plt.plot(data[i][0], data[i][1])
187     plt.xlabel('Moves')
188     plt.ylabel('Shannonian Entropy')
189     plt.title('Entropy over Time in Chess')
190     plt.savefig(f"figure-shannon-{dt_string}.png")
191     plt.show()
192
193 if __name__ == "__main__":
194     main()
```

```
1 import os, sys
2
3 def main():
4     baseName = sys.argv[1][0:-4]
5     count = 0
6     numFiles = 0
7     gameWriter = []
8     with open(sys.argv[1], 'r') as masterFile:
9         lines = masterFile.readlines()
10        for index, line in enumerate(lines):
11            gameWriter.append(line)
12            if line == "\n":
13                count += 1
14            if count == 2:
15                numFiles += 1
16                newGameFile = open(f"{baseName}-game{numFiles}.pgn", 'w')
17                print(f"{baseName}-game{numFiles}.pgn")
18                for line in gameWriter:
19                    newGameFile.writelines(line)
20                newGameFile.close()
21                gameWriter.clear()
22                count = 0
23 main()
24
```

```
1  #!/usr/bin/env python3
2  import sys, os
3  import numpy as np
4  import chess.pgn
5  import math
6  import matplotlib.pyplot as plt
7  import pandas as pd
8  from datetime import datetime
9
10 numFiles = len(sys.argv)
11 iterableFiles = range(1, numFiles)
12 baseName = [sys.argv[i][0:-4] for i in iterableFiles]
13 print(baseName)
14
15 def main():
16     data = {}
17     for i in iterableFiles:
18         with open(sys.argv[i]) as pgn:
19             game = chess.pgn.read_game(pgn)
20             board = game.board()
21             for moveNum, move in enumerate(game.mainline_moves()):
22                 legalMoves = list(board.legal_moves)
23                 numLegalMoves = len(list(board.legal_moves))
24                 stateSVG = open(f"{baseName[i-1]}-move{moveNum}gameState.svg", 'w')
25                 stateSVG.write(chess.svg.board(board, size = 350))
26                 stateSVG.close()
27                 board.push(move)
28
29 main()
```

```
1  #!/usr/bin/env python3
2  import sys, os
3  import re
4  from datetime import datetime
5
6  iterableFiles = (1, len(sys.argv))
7
8  def main():
9      whiteFile = open("White-Winning-Games.txt", 'w')
10     blackFile = open("Black-Winning-Games.txt", 'w')
11     drawFile = open("Draw-Games.txt", 'w')
12     blackSearch = re.compile(r"0-1")
13     whiteSearch = re.compile(r"1-1")
14     drawSearch = re.compile(r"1/2-1/2")
15     for i in iterableFiles:
16         with open(sys.argv[i], 'r') as pgn:
17             content = pgn.read()
18             whiteMatches = len(blackSearch.findall(content))
19             blackMatches = len(whiteSearch.findall(content))
20             drawMatches = len(drawSearch.findall(content))
21             if whiteMatches == 1:
22                 whiteFile.write(f"{sys.argv[i]}\n")
23             if blackMatches == 1:
24                 blackFile.write(f"{sys.argv[i]}\n")
25             if drawMatches == 1:
26                 drawFile.write(f"{sys.argv[i]}\n")
27     whiteFile.close()
28     blackFile.close()
29     drawFile.close()
30
31
32 main()
```

```

#!/usr/bin/env python3
import sys, os
import numpy as np
import chess.pgn
import math
import itertools
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime
import numpy.polynomial.polynomial as poly

#Up There are the dependencies exluding re, sys, and os the rest are standard
#Here is where the pgn files are from https://theweekinchess.com/a-year-of-pgn-game-files

numFiles = len(sys.argv)
iterableFiles = range(1, numFiles)
baseName = [sys.argv[i][0:-4] for i in iterableFiles]
log2 = lambda a: math.log(a, 2)

def main():
    #This is the time which is the name of each file so they are unquie each time
    runtime = datetime.now()
    dt_string = runtime.strftime("%d-%H-%M-%S")
    #the data we graph is in a dictionary
    data = {}
    #These go in the dictionary as a list of lists
    totalMoves = []
    totalShannonEntropies = []
    totalConditionalEntropies = []
    totalJointEntropy = []
    #open our data file which we write our statistics to
    dataFile = open(f"run-data-{dt_string}.out", 'w')
    #This iterates over all of the pgn files
    for i in iterableFiles:
        with open(sys.argv[i]) as pgn:
            #reads in the pgn and makes a chess board
            game = chess.pgn.read_game(pgn)
            board = game.board()
            dataFile.write(f"Game {i} and File Name {baseName[i-1]}\n")
            for moveNum, move in enumerate(game.mainline_moves()):
                if moveNum % 2 == 0:
                    #This identifies the piece on the square we move t
                    moveTo = (str(move)[2::])
                    if len(moveTo) == 3:
                        moveTo = moveTo[:-1]
                    piece = board.piece_at(chess.parse_square(moveTo))
                    #calculates the number of moves and the list of moves
                    legalMoves = list(board.legal_moves)
                    numLegalMoves = len(list(board.legal_moves))
                    #writes the data alternating white and then black
                    #pushes the board state to the next one
                    board.push(move)
                    #calculates the new set of moves for conditional entropy
                    newNumMovesCon = len(list(board.legal_moves))
                    dataFile.write(f"Player: White Move Number: {moveNum/2} Move: {move} Piece: {piece} Number of Legal Moves: {numLegalMoves} Entropy: {shannonEntropy(numLegalMoves)} Conditional E
                    #append the moves and entropies to our data
                    totalMoves.append(moveNum/2)
                    totalShannonEntropies.append(shannonEntropy(numLegalMoves))
                    totalConditionalEntropies.append(conditionalEntropy(numLegalMoves, newNumMovesCon))
                    totalJointEntropy.append(jointEntropy(numLegalMoves, newNumMovesCon))
                    #updates the data with the data filename:[stats, stats, stats]
                else:
                    board.push(move)
            data.update((baseName[i-1]: [totalMoves, totalShannonEntropies, totalConditionalEntropies, totalJointEntropy]))
            #empties the lists so we can fill and repeat with the next pgn file
            totalMoves = []
            totalShannonEntropies = []
            totalConditionalEntropies = []
            totalJointEntropy = []
            print(f"File {i} calculations completed")
        #close the file
        dataFile.close()
    #plots different junk
    plotShannonEntropy(data, dt_string)
    plotBestFitShannonEntropy(data, dt_string)
    plotConditionalEntropy(data, dt_string)
    plotBestFitConditionalEntropy(data, dt_string)
    plotJointEntropy(data, dt_string)
    plotBestJointEntropy(data, dt_string)

#!/usr/bin/env python3
import sys, os
import numpy as np
import chess.pgn
import math
import itertools
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime

#Up There are the dependencies exluding re, sys, and os the rest are standard
#Here is where the pgn files are from https://theweekinchess.com/a-year-of-pgn-game-files

numFiles = len(sys.argv)
iterableFiles = range(1, numFiles)
baseName = [sys.argv[i][0:-4] for i in iterableFiles]
log2 = lambda a: math.log(a, 2)

def main():
    #This is the time which is the name of each file so they are unquie each time
    runtime = datetime.now()
    dt_string = runtime.strftime("%d-%H-%M-%S")
    #the data we graph is in a dictionary
    data = {}
    #These go in the dictionary as a list of lists
    totalMoves = []

```

```

105 totalMoves = []
106 totalShannonEntropies = []
107 totalConditionalEntropies = []
108 totalJointEntropy = []
109 #open our data file which we write our statistics to
110 dataFile = open(f"run-data-{dt_string}.out", 'w')
111 #This iterates over all of the pgn files
112 for i in iterableFiles:
113     with open(sys.argv[i]) as pgn:
114         #reads in the pgn and makes a chess board
115         game = chess.pgn.read_game(pgn)
116         board = game.board()
117         dataFile.write(f"Game {i} and File Name {baseName[i-1]}\n")
118         for moveNum, move in enumerate(game.mainline_moves()):
119             if moveNum % 2 != 0:
120                 #This identifies the piece on the square we move t
121                 moveTo = (str(move)[2::])
122                 if len(moveTo) == 3:
123                     moveTo = moveTo[:-1]
124                     piece = board.piece_at(chess.parse_square(moveTo))
125                     #calculates the number of moves and the list of moves
126                     legalMoves = list(board.legal_moves)
127                     numLegalMoves = len(list(board.legal_moves))
128                     #writes the data alternating white and then black
129                     #pushes the board state to the next one
130                     board.push(move)
131                     #calculates the new set of moves for conditional entropy
132                     newNumMovesCon = len(list(board.legal_moves))
133                     dataFile.write(f"Player: Black Move Number: {moveNum // 2} Move: {move} Piece: {piece} Number of Legal Moves: {numLegalMoves} Entropy: {shannonEntropy(numLegalMoves)} Conditional
134                     #append the moves and entropies to our data
135                     totalMoves.append(moveNum//2)
136                     totalShannonEntropies.append(shannonEntropy(numLegalMoves))
137                     totalConditionalEntropies.append(conditionalEntropy(numLegalMoves, newNumMovesCon))
138                     totalJointEntropy.append(jointEntropy(numLegalMoves, newNumMovesCon))
139                     #updates the data with the data filename:[stats, stats, stats]
140                 else:
141                     board.push(move)
142             data.update(({baseName[i-1]: [totalMoves, totalShannonEntropies, totalConditionalEntropies, totalJointEntropy]}))
143             #empties the lists so we can fill and repeat with the next pgn file
144             totalMoves = []
145             totalShannonEntropies = []
146             totalConditionalEntropies = []
147             totalJointEntropy = []
148         #close the file
149         dataFile.close()
150         #plots different junk
151         plotShannonEntropy(data, dt_string)
152         plotBestFitShannonEntropy(data, dt_string)
153         plotConditionalEntropy(data, dt_string)
154         plotBestFitConditionalEntropy(data, dt_string)
155         plotJointEntropy(data, dt_string)
156         plotBestJointEntropy(data, dt_string)
157
158 #!/usr/bin/env python3
159 import sys, os
160 import numpy as np
161 import chess.pgn
162 import math
163 import itertools
164 import matplotlib.pyplot as plt
165 import pandas as pd
166 from datetime import datetime
167 #Up There are the dependencies excluding re, sys, and os the rest are standard
168 #Here is where the pgn files are from https://theweekinchess.com/a-year-of-pgn-game-files
169
170 numFiles = len(sys.argv)
171 iterableFiles = range(1, numFiles)
172 baseName = [sys.argv[i][0:-4] for i in iterableFiles]
173 log2 = lambda a: math.log(a, 2)
174
175 def main():
176     #This is the time which is the name of each file so they are unqiue each time
177     runtime = datetime.now()
178     dt_string = runtime.strftime("%d-%H-%M-%S")
179     #the data we graph is in a dictionary
180     data = {}
181     #These go in the dictionary as a list of lists
182     totalMoves = []
183     totalShannonEntropies = []
184     totalConditionalEntropies = []
185     totalJointEntropy = []
186     #open our data file which we write our statistics to
187     dataFile = open(f"run-data-{dt_string}.out", 'w')
188     #This iterates over all of the pgn files
189     for i in iterableFiles:
190         with open(sys.argv[i]) as pgn:
191             #reads in the pgn and makes a chess board
192             game = chess.pgn.read_game(pgn)
193             board = game.board()
194             dataFile.write(f"Game {i} and File Name {baseName[i-1]}\n")
195             for moveNum, move in enumerate(game.mainline_moves()):
196                 if moveNum % 2 != 0:
197                     #This identifies the piece on the square we move t
198                     moveTo = (str(move)[2::])
199                     if len(moveTo) == 3:
200                         moveTo = moveTo[:-1]
201                         piece = board.piece_at(chess.parse_square(moveTo))
202                         #calculates the number of moves and the list of moves
203                         legalMoves = list(board.legal_moves)
204                         numLegalMoves = len(list(board.legal_moves))
205                         #writes the data alternating white and then black
206                         #pushes the board state to the next one
207                         board.push(move)
208                         #calculates the new set of moves for conditional entropy
209                         newNumMovesCon = len(list(board.legal_moves))
210                         dataFile.write(f"Player: Black Move Number: {moveNum // 2} Move: {move} Piece: {piece} Number of Legal Moves: {numLegalMoves} Entropy: {shannonEntropy(numLegalMoves)} Conditional
211                         #append the moves and entropies to our data

```

```

212         #append the moves and entropies to our data
213         totalMoves.append(moveNum//2)
214         totalShannonEntropies.append(shannonEntropy(numLegalMoves))
215         totalConditionalEntropies.append(conditionalEntropy(numLegalMoves, newNumMovesCon))
216         totalJointEntropy.append(jointEntropy(numLegalMoves, newNumMovesCon))
217         #updates the data with the data filename:[stats, stats, stats]
218     else:
219         board.push(move)
220         data.update({baseName[i-1]: [totalMoves, totalShannonEntropies, totalConditionalEntropies, totalJointEntropy]})
221         #empties the lists so we can fill and repeat with the next pgn file
222         totalMoves = []
223         totalShannonEntropies = []
224         totalConditionalEntropies = []
225         totalJointEntropy = []
226     #close the file
227     dataFile.close()
228     #plots different junk
229     plotShannonEntropy(data, dt_string)
230     plotBestFitShannonEntropy(data, dt_string)
231     plotConditionalEntropy(data, dt_string)
232     plotBestFitConditionalEntropy(data, dt_string)
233     plotJointEntropy(data, dt_string)
234     plotBestJointEntropy(data, dt_string)
235
236
237
238 def plotBestJointEntropy(data, dt_string):
239     bestFitXs = []
240     bestFitYs = []
241     plt.figure()
242     for i in list(data.keys()):
243         plt.scatter(data[i][0], data[i][3])
244         bestFitXs.append(data[i][0])
245         bestFitYs.append(data[i][3])
246     bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
247     bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
248     model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
249     print(model)
250     nModel = np.polyder(model)
251     print(nModel)
252     plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
253     plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
254     plt.legend(handlelength=3)
255     plt.xlabel('Moves')
256     plt.ylabel('Joint Entropy (bits)')
257     plt.title('Joint Entropy During the 2021 World Chess Championships', fontsize = 20)
258     plt.savefig(f"figure-joint-best-{dt_string}.png")
259     plt.show()
260
261
262 def plotJointEntropy(data, dt_string):
263     plt.figure()
264     for i in list(data.keys()):
265         plt.plot(data[i][0], data[i][3])
266
267     plt.xlabel('Moves')
268     plt.ylabel('Joint Entropy')
269     plt.title('Entropy over Time in Chess')
270     plt.savefig(f"figure-joint-{dt_string}.png")
271     plt.show()
272
273
274 def conditionalEntropy(probabilities, newProbabilities):
275     conditionalEntropy = -1 * sum(conditionalEntropy := [(1/probabilities)^(1/newProbabilities) * log2(((1/probabilities)^(1/newProbabilities)))/((1/probabilities))]) for pX in range(1, probabilities+1) for pY in range(1, newProbabilities+1))
276     return conditionalEntropy
277
278
279 def shannonEntropy(probabilities):
280     entropy = -1 * sum(entropies := [((1/probabilities)*log2(1/probabilities)) for prob in range(1, probabilities+1)])
281     return entropy
282
283
284 def jointEntropy(probabilities, newProbabilities):
285     jointEntropy = -1 * sum(jointEntropy := [(1/probabilities) * (1/newProbabilities) * log2((1/probabilities)^(1/newProbabilities))]) for pX in range(1, probabilities+1) for pY in range(1, newProbabilities+1))
286     return jointEntropy
287
288
289
290 def plotConditionalEntropy(data, dt_string):
291     plt.figure()
292     for i in list(data.keys()):
293         plt.plot(data[i][0], data[i][2])
294     plt.xlabel('Moves')
295     plt.ylabel('Conditional Entropy (bits)')
296     plt.title('Entropy over Time in Chess')
297     plt.savefig(f"figure-conditional-{dt_string}.png")
298     plt.show()
299
300
301 def plotBestFitConditionalEntropy(data, dt_string):
302     bestFitXs = []
303     bestFitYs = []
304     plt.figure()
305     for i in list(data.keys()):
306         plt.scatter(data[i][0], data[i][2])
307         bestFitXs.append(data[i][0])
308         bestFitYs.append(data[i][2])
309     bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
310     bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
311     model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
312     print(model)
313     nModel = np.polyder(model)
314     print(nModel)
315     plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
316     plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
317     plt.legend(handlelength=3)
318     plt.xlabel('Moves')
319     plt.ylabel('Conditional Entropy (bits)')
320     plt.title('Conditional Entropy During the 2021 World Chess Championships', fontsize = 20)
321     plt.savefig(f"figure-conditional-best-{dt_string}.png")
322     plt.show()
323
324
325 def plotBestFitShannonEntropy(data, dt_string):

```

```

320 bestFitXs = []
321 bestFitYs = []
322 plt.figure()
323 for i in list(data.keys()):
324     plt.scatter(data[i][0], data[i][1])
325     bestFitXs.append(data[i][0])
326     bestFitYs.append(data[i][1])
327 bestFitXs = np.array(list(itertools.chain("bestFitXs")))
328 bestFitYs = np.array(list(itertools.chain("bestFitYs")))
329 model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
330 print(model)
331 nModel = np.polyder(model)
332 print(nModel)
333 plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
334 plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
335 plt.legend(handlelength=3)
336 plt.xlabel('Moves')
337 plt.ylabel('Shannonian Entropy (bits)')
338 plt.title('Shannon Entropy During the 2021 World Chess Championships', fontsize = 20)
339 plt.savefig("figure-shannon-best-{dt_string}.png")
340 plt.show()
341
342 def plotShannonEntropy(data, dt_string):
343     plt.figure()
344     for i in list(data.keys()):
345         plt.plot(data[i][0], data[i][1])
346         plt.xlabel('Moves')
347         plt.ylabel('Shannonian Entropy')
348         plt.title('Entropy over Time in Chess')
349         plt.savefig("figure-shannon-{dt_string}.png")
350         plt.show()

if __name__ == "__main__":
    main()

```