```python
#!/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 E
                    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]})
```

```
            #data.update({baseName[0] : [totalMoves, totalShannonEntropies, totalConditionalEntropies]})
            #empties the lists so we can fill and repeat with the next pgn file
            totalMoves = []
            totalShannonEntropies = []
            totalConditionalEntropies = []
            totalJointEntropy = []
            print(i)
    engine.quit()
    dataFile.close()
    plotShannonEntropy(data, dt_string)
    plotBestFitShannonEntropy(data, dt_string)
    plotConditionalEntropy(data, dt_string)
    plotBestFitConditionalEntropy(data, dt_string)
    plotJointEntropy(data, dt_string)
    plotBestJointEntropy(data, dt_string)
    exit()

def conditionalEntropy(probabilities, newProbabilities):
    conditionalEntropy = -1 * sum(conditionalEntropy := [pX*pY * (log2(pX)/(pX*pY)) for pX in probabilities for pY in newProbabilities if pY != 0 and pX != 0])
    return conditionalEntropy

def shannonEntropy(probabilities):
    entropy = -1 * sum(entropies := [((prob)*log2(prob)) for prob in probabilities if prob != 0])
    return entropy

def jointEntropy(probabilities, newProbabilities):
    jointEntropy = -1 * sum((pX * pY * log2(pX*pY)) for pX in probabilities for pY in newProbabilities if pX !=0 and pY != 0)
    return jointEntropy



def plotConditionalEntropy(data, dt_string):
    plt.figure()
    for i in list(data.keys()):
        plt.plot(data[i][0], data[i][2])
    plt.xlabel('Moves')
    plt.ylabel('Conditional Entropy (bits)')
    plt.title('Entropy over Time in Chess')
    plt.savefig(f"figure-conditional-{dt_string}.png")
    plt.show()

def plotBestFitConditionalEntropy(data, dt_string):
    bestFitXs = []
    bestFitYs = []
    plt.figure()
    for i in list(data.keys()):
        plt.scatter(data[i][0], data[i][2])
        bestFitXs.append(data[i][0])
        bestFitYs.append(data[i][2])
    bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
    bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
    model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
    print(model)
    nModel = np.polyder(model)
    print(nModel)
    plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
    plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
    plt.legend(handlelength=3)
    plt.xlabel('Moves')
    plt.ylabel('Conditional Entropy  (bits)')
    plt.title('Conditional Entropy During the 2021 World Chess Championships', fontsize = 20)
    plt.savefig(f"figure-conditional-best-{dt_string}.png")
    plt.show()

def plotBestFitShannonEntropy(data, dt_string):
    bestFitXs = []
    bestFitYs = []
    plt.figure()
    for i in list(data.keys()):
        plt.scatter(data[i][0], data[i][1])
        bestFitXs.append(data[i][0])
        bestFitYs.append(data[i][1])
    bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
    bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
    model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
    print(model)
    nModel = np.polyder(model)
    print(nModel)
    plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
    plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
    plt.legend(handlelength=3)
    plt.xlabel('Moves')
    plt.ylabel('Shannonian Entropy (bits)')
    plt.title('Shannon Entropy During the 2021 World Chess Championships',  fontsize = 20)
    plt.savefig(f"figure-shannon-best-{dt_string}.png")
    plt.show()

def plotShannonEntropy(data, dt_string):
    plt.figure()
    for i in list(data.keys()):
        plt.plot(data[i][0], data[i][1])
    plt.xlabel('Moves')
    plt.ylabel('Shannonian Entropy')
    plt.title('Entropy over Time in Chess')
    plt.savefig(f"figure-shannon-{dt_string}.png")
    plt.show()


def plotBestJointEntropy(data, dt_string):
    bestFitXs = []
    bestFitYs = []
    plt.figure()
    for i in list(data.keys()):
        plt.scatter(data[i][0], data[i][3])
        bestFitXs.append(data[i][0])
```

```
        bestFitYs.append(data[i][3])
    bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
    bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
    model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
    print(model)
    nModel = np.polyder(model)
    print(nModel)
    plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
    plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
    plt.legend(handlelength=3)
    plt.xlabel('Moves')
    plt.ylabel('Joint Entropy (bits)')
    plt.title('Joint Entropy During the 2021 World Chess Championships', fontsize = 20)
    plt.savefig(f"figure-joint-best-{dt_string}.png")
    plt.show()

def plotJointEntropy(data, dt_string):
    plt.figure()
    for i in list(data.keys()):
        plt.plot(data[i][0], data[i][3])
    plt.xlabel('Moves')
    plt.ylabel('Joint Entropy')
    plt.title('Entropy over Time in Chess')
    plt.savefig(f"figure-joint-{dt_string}.png")
    plt.show()


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

```python
#!/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 and moveNum != 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.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]
                    # print("Weights ", weights)
                    # print("Probabilities", newProbabilities)
                    # print("Board Push")
                    entropy = shannonEntropy(probabilities)
                    conEntropy = conditionalEntropy(probabilities, newProbabilities)
                    jEntropy = jointEntropy(probabilities, newProbabilities)
                    dataFile.write(f"Player: Black Move Number: {moveNum} Move: {move} Piece: {piece} Number of Better Moves: {numBetterMoves} Entropy: {entropy} Conditional Entropy: {conEntropy} Joint E
                    totalMoves.append(moveNum)
                    totalShannonEntropies.append(shannonEntropy(probabilities))
                    totalConditionalEntropies.append(conEntropy)
                    totalJointEntropy.append(jEntropy)
                else:
                    #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")
                entropy = shannonEntropy(probabilities)
                conEntropy = conditionalEntropy(probabilities, newProbabilities)
                jEntropy = jointEntropy(probabilities, newProbabilities)
                dataFile.write(f"Player: White Move Number: {moveNum} Move: {move} Piece: {piece} Number of Better Moves: {numBetterMoves} Entropy: {entropy} Conditional Entropy: {conEntropy} Joint E
                totalMoves.append(moveNum)
                totalShannonEntropies.append(shannonEntropy(probabilities))
                totalConditionalEntropies.append(conEntropy)
                totalJointEntropy.append(jEntropy)
            #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]})
            #data.update({baseName[0] : [totalMoves, totalShannonEntropies, totalConditionalEntropies]})
            #empties the lists so we can fill and repeat with the next pgn file
            totalMoves = []
            totalShannonEntropies = []
            totalConditionalEntropies = []
            totalJointEntropy = []
            print(f"File completion {i}")
    engine.quit()
    dataFile.close()
    plotShannonEntropy(data, dt_string)
    plotBestFitShannonEntropy(data, dt_string)
    plotConditionalEntropy(data, dt_string)
    plotBestFitConditionalEntropy(data, dt_string)
    plotJointEntropy(data, dt_string)
    plotBestJointEntropy(data, dt_string)
    exit()


def conditionalEntropy(probabilities, newProbabilities):
    conditionalEntropy = -1 * sum(conditionalEntropy := [pX*pY * (log2(pX)/(pX*pY)) for pX in probabilities for pY in newProbabilities if pY != 0 and pX != 0])
    return conditionalEntropy

def shannonEntropy(probabilities):
    entropy = -1 * sum(entropies := [((prob)*log2(prob)) for prob in probabilities if prob != 0])
    return entropy

def jointEntropy(probabilities, newProbabilities):
    jointEntropy = -1 * sum((pX * pY * log2(pX*pY)) for pX in probabilities for pY in newProbabilities if pX !=0 and pY != 0)
    return jointEntropy



def plotConditionalEntropy(data, dt_string):
    plt.figure()
    for i in list(data.keys()):
        plt.plot(data[i][0], data[i][2])
    plt.xlabel('Moves')
    plt.ylabel('Conditional Entropy (bits)')
    plt.title('Entropy over Time in Chess')
    plt.savefig(f"figure-conditional-{dt_string}.png")
    plt.show()

def plotBestFitConditionalEntropy(data, dt_string):
    bestFitXs = []
    bestFitYs = []
    plt.figure()
    for i in list(data.keys()):
        plt.scatter(data[i][0], data[i][2])
        bestFitXs.append(data[i][0])
        bestFitYs.append(data[i][2])
    bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
```

```
    bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
    model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
    print(model)
    nModel = np.polyder(model)
    print(nModel)
    plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
    plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
    plt.legend(handlelength=3)
    plt.xlabel('Moves')
    plt.ylabel('Conditional Entropy  (bits)')
    plt.title('Conditional Entropy During the 2021 World Chess Championships', fontsize = 20)
    plt.savefig(f"figure-conditional-best-{dt_string}.png")
    plt.show()

def plotBestFitShannonEntropy(data, dt_string):
    bestFitXs = []
    bestFitYs = []
    plt.figure()
    for i in list(data.keys()):
        plt.scatter(data[i][0], data[i][1])
        bestFitXs.append(data[i][0])
        bestFitYs.append(data[i][1])
    bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
    bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
    model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
    print(model)
    nModel = np.polyder(model)
    print(nModel)
    plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
    plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
    plt.legend(handlelength=3)
    plt.xlabel('Moves')
    plt.ylabel('Shannonian Entropy (bits)')
    plt.title('Shannon Entropy During the 2021 World Chess Championships',  fontsize = 20)
    plt.savefig(f"figure-shannon-best-{dt_string}.png")
    plt.show()

def plotShannonEntropy(data, dt_string):
    plt.figure()
    for i in list(data.keys()):
        plt.plot(data[i][0], data[i][1])
    plt.xlabel('Moves')
    plt.ylabel('Shannonian Entropy')
    plt.title('Entropy over Time in Chess')
    plt.savefig(f"figure-shannon-{dt_string}.png")
    plt.show()


def plotBestJointEntropy(data, dt_string):
    bestFitXs = []
    bestFitYs = []
    plt.figure()
    for i in list(data.keys()):

        plt.scatter(data[i][0], data[i][3])
        bestFitXs.append(data[i][0])
        bestFitYs.append(data[i][3])
    bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
    bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
    model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
    print(model)
    nModel = np.polyder(model)
    print(nModel)
    plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
    plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
    plt.legend(handlelength=3)
    plt.xlabel('Moves')
    plt.ylabel('Joint Entropy (bits)')
    plt.title('Joint Entropy During the 2021 World Chess Championships', fontsize = 20)
    plt.savefig(f"figure-joint-best-{dt_string}.png")
    plt.show()

def plotJointEntropy(data, dt_string):
    plt.figure()
    for i in list(data.keys()):
        plt.plot(data[i][0], data[i][3])
    plt.xlabel('Moves')
    plt.ylabel('Joint Entropy')
    plt.title('Entropy over Time in Chess')
    plt.savefig(f"figure-joint-{dt_string}.png")
    plt.show()


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

```python
#!/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))

                    # 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(f"Player: White Move Number: {whiteMove} Move: {move} Piece: {piece} Number of Better Moves: {numBetterMoves} Entropy: {entropy} Conditional Entropy: {conEntropy} Joint
                    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]})
```

```
        #data.update({baseName[0] : [totalMoves, totalShannonEntropies, totalConditionalEntropies]})
        #empties the lists so we can fill and repeat with the next pgn file
        totalMoves = []
        totalShannonEntropies = []
        totalConditionalEntropies = []
        totalJointEntropy = []
        print(i)
    engine.quit()
    dataFile.close()
    plotShannonEntropy(data, dt_string)
    plotBestFitShannonEntropy(data, dt_string)
    plotConditionalEntropy(data, dt_string)
    plotBestFitConditionalEntropy(data, dt_string)
    plotJointEntropy(data, dt_string)
    plotBestJointEntropy(data, dt_string)
    exit()


def conditionalEntropy(probabilities, newProbabilities):
    conditionalEntropy = -1 * sum(conditionalEntropy := [pX*pY * (log2(pX)/(pX*pY)) for pX in probabilities for pY in newProbabilities if pY != 0 and pX != 0])
    return conditionalEntropy

def shannonEntropy(probabilities):
    entropy = -1 * sum(entropies := [((prob)*log2(prob)) for prob in probabilities if prob != 0])
    return entropy

def jointEntropy(probabilities, newProbabilities):
    jointEntropy = -1 * sum((pX * pY * log2(pX*pY)) for pX in probabilities for pY in newProbabilities if pX !=0 and pY != 0)
    return jointEntropy



def plotConditionalEntropy(data, dt_string):
    plt.figure()
    for i in list(data.keys()):
        plt.plot(data[i][0], data[i][2])
    plt.xlabel('Moves')
    plt.ylabel('Conditional Entropy (bits)')
    plt.title('Entropy over Time in Chess')
    plt.savefig(f"figure-conditional-{dt_string}.png")
    plt.show()

def plotBestFitConditionalEntropy(data, dt_string):
    bestFitXs = []
    bestFitYs = []
    plt.figure()
    for i in list(data.keys()):
        plt.scatter(data[i][0], data[i][2])
        bestFitXs.append(data[i][0])
        bestFitYs.append(data[i][2])
    bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
    bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
    model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
    print(model)
    nModel = np.polyder(model)
    print(nModel)
    plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
    plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
    plt.legend(handlelength=3)
    plt.xlabel('Moves')
    plt.ylabel('Conditional Entropy  (bits)')
    plt.title('Conditional Entropy During the 2021 World Chess Championships', fontsize = 20)
    plt.savefig(f"figure-conditional-best-{dt_string}.png")
    plt.show()

def plotBestFitShannonEntropy(data, dt_string):
    bestFitXs = []
    bestFitYs = []
    plt.figure()
    for i in list(data.keys()):
        plt.scatter(data[i][0], data[i][1])
        bestFitXs.append(data[i][0])
        bestFitYs.append(data[i][1])
    bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
    bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
    model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
    print(model)
    nModel = np.polyder(model)
    print(nModel)
    plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
    plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
    plt.legend(handlelength=3)
    plt.xlabel('Moves')
    plt.ylabel('Shannonian Entropy (bits)')
    plt.title('Shannon Entropy During the 2021 World Chess Championships',  fontsize = 20)
    plt.savefig(f"figure-shannon-best-{dt_string}.png")
    plt.show()

def plotShannonEntropy(data, dt_string):
    plt.figure()
    for i in list(data.keys()):
        plt.plot(data[i][0], data[i][1])
    plt.xlabel('Moves')
    plt.ylabel('Shannonian Entropy')
    plt.title('Entropy over Time in Chess')
    plt.savefig(f"figure-shannon-{dt_string}.png")
    plt.show()


def plotBestJointEntropy(data, dt_string):
    bestFitXs = []
    bestFitYs = []
    plt.figure()
    for i in list(data.keys()):
        plt.scatter(data[i][0], data[i][3])
```

```
        bestFitXs.append(data[i][0])
        bestFitYs.append(data[i][3])
    bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
    bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
    model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
    print(model)
    nModel = np.polyder(model)
    print(nModel)
    plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
    plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
    plt.legend(handlelength=3)
    plt.xlabel('Moves')
    plt.ylabel('Joint Entropy (bits)')
    plt.title('Joint Entropy During the 2021 World Chess Championships', fontsize = 20)
    plt.savefig(f"figure-joint-best-{dt_string}.png")
    plt.show()

def plotJointEntropy(data, dt_string):
    plt.figure()
    for i in list(data.keys()):
        plt.plot(data[i][0], data[i][3])
    plt.xlabel('Moves')
    plt.ylabel('Joint Entropy')
    plt.title('Entropy over Time in Chess')
    plt.savefig(f"figure-joint-{dt_string}.png")
    plt.show()


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

```python
#!/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 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
    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: 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 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
    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: 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)



def plotBestJointEntropy(data, dt_string):
    bestFitXs = []
    bestFitYs = []
    plt.figure()
    for i in list(data.keys()):
        plt.scatter(data[i][0], data[i][3])
        bestFitXs.append(data[i][0])
        bestFitYs.append(data[i][3])
    bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
    bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
    model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
    print(model)
    nModel = np.polyder(model)
    print(nModel)
    plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
    plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
    plt.legend(handlelength=3)
    plt.xlabel('Moves')
    plt.ylabel('Joint Entropy (bits)')
    plt.title('Joint Entropy During the 2021 World Chess Championships', fontsize = 20)
    plt.savefig(f"figure-joint-best-{dt_string}.png")
    plt.show()

def plotJointEntropy(data, dt_string):
    plt.figure()
    for i in list(data.keys()):
        plt.plot(data[i][0], data[i][3])
    plt.xlabel('Moves')
    plt.ylabel('Joint Entropy')
    plt.title('Entropy over Time in Chess')
    plt.savefig(f"figure-joint-{dt_string}.png")
    plt.show()

def conditionalEntropy(probabilities, newProbabilities):
    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
    return conditionalEntropy

def shannonEntropy(probabilities):
    entropy = -1 * sum(entropies := [((1/probabilities)*log2(1/probabilities)) for prob in range(1, probabilities+1)])
    return entropy


def jointEntropy(probabilities, newProbabilities):
    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))
    return jointEntropy



def plotConditionalEntropy(data, dt_string):
    plt.figure()
    for i in list(data.keys()):
        plt.plot(data[i][0], data[i][2])
    plt.xlabel('Moves')
    plt.ylabel('Conditional Entropy (bits)')
    plt.title('Entropy over Time in Chess')
    plt.savefig(f"figure-conditional-{dt_string}.png")
```

```
    plt.show()

def plotBestFitConditionalEntropy(data, dt_string):
    bestFitXs = []
    bestFitYs = []
    plt.figure()
    for i in list(data.keys()):
        plt.scatter(data[i][0], data[i][2])
        bestFitXs.append(data[i][0])
        bestFitYs.append(data[i][2])
    bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
    bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
    model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
    print(model)
    nModel = np.polyder(model)
    print(nModel)
    plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
    plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
    plt.legend(handlelength=3)
    plt.xlabel('Moves')
    plt.ylabel('Conditional Entropy  (bits)')
    plt.title('Conditional Entropy During the 2021 World Chess Championships', fontsize = 20)
    plt.savefig(f"figure-conditional-best-{dt_string}.png")
    plt.show()

def plotBestFitShannonEntropy(data, dt_string):
    bestFitXs = []
    bestFitYs = []
    plt.figure()
    for i in list(data.keys()):
        plt.scatter(data[i][0], data[i][1])
        bestFitXs.append(data[i][0])
        bestFitYs.append(data[i][1])
    bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
    bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
    model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
    print(model)
    nModel = np.polyder(model)
    print(nModel)
    plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
    plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
    plt.legend(handlelength=3)
    plt.xlabel('Moves')
    plt.ylabel('Shannonian Entropy (bits)')
    plt.title('Shannon Entropy During the 2021 World Chess Championships',  fontsize = 20)
    plt.savefig(f"figure-shannon-best-{dt_string}.png")
    plt.show()

def plotShannonEntropy(data, dt_string):
    plt.figure()
    for i in list(data.keys()):
        plt.plot(data[i][0], data[i][1])
    plt.xlabel('Moves')
    plt.ylabel('Shannonian Entropy')
    plt.title('Entropy over Time in Chess')
    plt.savefig(f"figure-shannon-{dt_string}.png")
    plt.show()

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

```python
#!/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 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
    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()):
                #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))
                if moveNum % 2 == 0:
                    dataFile.write(f"Player: White Move Number: {moveNum} Move: {move} Piece: {piece} Number of Legal Moves: {numLegalMoves} Entropy: {shannonEntropy(numLegalMoves)} Conditional En
                else:
                    dataFile.write(f"Player: Black Move Number: {moveNum} Move: {move} Piece: {piece} Number of Legal Moves: {numLegalMoves} Entropy: {shannonEntropy(numLegalMoves)} Conditional Ent
                #append the moves and entropies to our data
                totalMoves.append(moveNum)
                totalShannonEntropies.append(shannonEntropy(numLegalMoves))
                totalConditionalEntropies.append(conditionalEntropy(numLegalMoves, newNumMovesCon))
                totalJointEntropy.append(jointEntropy(numLegalMoves, newNumMovesCon))
        #updates the data with the data filename:[stats, stats, stats]
        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} completion")
    #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)


def plotBestJointEntropy(data, dt_string):
    bestFitXs = []
    bestFitYs = []
    plt.figure()
    for i in list(data.keys()):
        plt.scatter(data[i][0], data[i][3])
        bestFitXs.append(data[i][0])
        bestFitYs.append(data[i][3])
    bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
    bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
    model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
    print(model)
    nModel = np.polyder(model)
    print(nModel)
    plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
    plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
    plt.legend(handlelength=3)
    plt.xlabel('Moves')
    plt.ylabel('Joint Entropy (bits)')
    plt.title('Joint Entropy During the 2021 World Chess Championships', fontsize = 20)
    plt.savefig(f"figure-joint-best-{dt_string}.png")
    plt.show()

def plotJointEntropy(data, dt_string):
    plt.figure()
    for i in list(data.keys()):
```

```
        plt.plot(data[i][0], data[i][3])
    plt.xlabel('Moves')
    plt.ylabel('Joint Entropy')
    plt.title('Entropy over Time in Chess')
    plt.savefig(f"figure-joint-{dt_string}.png")
    plt.show()

def conditionalEntropy(probabilities, newProbabilities):
    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
    return conditionalEntropy

def shannonEntropy(probabilities):
    entropy = -1 * sum(entropies := [((1/probabilities)*log2(1/probabilities)) for prob in range(1, probabilities+1)])
    return entropy

def jointEntropy(probabilities, newProbabilities):
    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))
    return jointEntropy


def plotConditionalEntropy(data, dt_string):
    plt.figure()
    for i in list(data.keys()):
        plt.plot(data[i][0], data[i][2])
    plt.xlabel('Moves')
    plt.ylabel('Conditional Entropy (bits)')
    plt.title('Entropy over Time in Chess')
    plt.savefig(f"figure-conditional-{dt_string}.png")
    plt.show()

def plotBestFitConditionalEntropy(data, dt_string):
    bestFitXs = []
    bestFitYs = []
    plt.figure()
    for i in list(data.keys()):
        plt.scatter(data[i][0], data[i][2])
        bestFitXs.append(data[i][0])
        bestFitYs.append(data[i][2])
    bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
    bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
    model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
    print(model)
    nModel = np.polyder(model)
    print(nModel)
    plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
    plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
    plt.legend(handlelength=3)
    plt.xlabel('Moves')
    plt.ylabel('Conditional Entropy  (bits)')
    plt.title('Conditional Entropy During the 2021 World Chess Championships', fontsize = 20)
    plt.savefig(f"figure-conditional-best-{dt_string}.png")
    plt.show()

def plotBestFitShannonEntropy(data, dt_string):
    bestFitXs = []
    bestFitYs = []
    plt.figure()
    for i in list(data.keys()):
        plt.scatter(data[i][0], data[i][1])
        bestFitXs.append(data[i][0])
        bestFitYs.append(data[i][1])
    bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
    bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
    model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
    print(model)
    nModel = np.polyder(model)
    print(nModel)
    plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
    plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
    plt.legend(handlelength=3)
    plt.xlabel('Moves')
    plt.ylabel('Shannonian Entropy (bits)')
    plt.title('Shannon Entropy During the 2021 World Chess Championships',  fontsize = 20)
    plt.savefig(f"figure-shannon-best-{dt_string}.png")
    plt.show()

def plotShannonEntropy(data, dt_string):
    plt.figure()
    for i in list(data.keys()):
        plt.plot(data[i][0], data[i][1])
    plt.xlabel('Moves')
    plt.ylabel('Shannonian Entropy')
    plt.title('Entropy over Time in Chess')
    plt.savefig(f"figure-shannon-{dt_string}.png")
    plt.show()

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

```
import os, sys

def main():
    baseName = sys.argv[1][0:-4]
    count = 0
    numFiles = 0
    gameWriter = []
    with open(sys.argv[1], 'r') as masterFile:
        lines = masterFile.readlines()
        for index, line in enumerate(lines):
            gameWriter.append(line)
            if line == "\n":
                count += 1
            if count == 2:
                numFiles += 1
                newGameFile = open(f"{baseName}-game{numFiles}.pgn", 'w')
                print(f"{baseName}-game{numFiles}.pgn")
                for line in gameWriter:
                    newGameFile.writelines(line)
                newGameFile.close()
                gameWriter.clear()
                count = 0
main()
```

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

numFiles = len(sys.argv)
iterableFiles = range(1, numFiles)
baseName = [sys.argv[i][0:-4] for i in iterableFiles]
print(baseName)

def main():
    data = {}
    for i in iterableFiles:
        with open(sys.argv[i]) as pgn:
            game = chess.pgn.read_game(pgn)
            board = game.board()
            for moveNum, move in enumerate(game.mainline_moves()):
                legalMoves = list(board.legal_moves)
                numLegalMoves = len(list(board.legal_moves))
                stateSVG = open(f"{baseName[i-1]}-move{moveNum}gameState.svg", 'w')
                stateSVG.write(chess.svg.board(board, size = 350))
                stateSVG.close()
                board.push(move)

main()
```

```python
#!/usr/bin/env python3
import sys, os
import re
from datetime import datetime

iterableFiles = (1, len(sys.argv))

def main():
    whiteFile = open("White-Winning-Games.txt", 'w')
    blackFile = open("Black-Winning-Games.txt", 'w')
    drawFile = open("Draw-Games.txt", 'w')
    blackSearch = re.compile(r"0-1")
    whiteSearch = re.compile(r"1-1")
    drawSearch = re.compile(r"1/2-1/2")
    for i in iterableFiles:
        with open(sys.argv[i], 'r') as pgn:
            content = pgn.read()
            whiteMatches = len(blackSearch.findall(content))
            blackMatches = len(whiteSearch.findall(content))
            drawMatches = len(drawSearch.findall(content))
            if whiteMatches == 1:
                whiteFile.write(f"{sys.argv[i]}\n")
            if blackMatches == 1:
                blackFile.write(f"{sys.argv[i]}\n")
            if drawMatches == 1:
                drawFile.write(f"{sys.argv[i]}\n")
    whiteFile.close()
    blackFile.close()
    drawFile.close()


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 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
    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 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
    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: 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 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
    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: 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
```

```
                #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)



def plotBestJointEntropy(data, dt_string):
    bestFitXs = []
    bestFitYs = []
    plt.figure()
    for i in list(data.keys()):
        plt.scatter(data[i][0], data[i][3])
        bestFitXs.append(data[i][0])
        bestFitYs.append(data[i][3])
    bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
    bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
    model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
    print(model)
    nModel = np.polyder(model)
    print(nModel)
    plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
    plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
    plt.legend(handlelength=3)
    plt.xlabel('Moves')
    plt.ylabel('Joint Entropy (bits)')
    plt.title('Joint Entropy During the 2021 World Chess Championships', fontsize = 20)
    plt.savefig(f"figure-joint-best-{dt_string}.png")
    plt.show()

def plotJointEntropy(data, dt_string):
    plt.figure()
    for i in list(data.keys()):
        plt.plot(data[i][0], data[i][3])

    plt.xlabel('Moves')
    plt.ylabel('Joint Entropy')
    plt.title('Entropy over Time in Chess')
    plt.savefig(f"figure-joint-{dt_string}.png")
    plt.show()

def conditionalEntropy(probabilities, newProbabilities):
    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
    return conditionalEntropy

def shannonEntropy(probabilities):
    entropy = -1 * sum(entropies := [((1/probabilities)*log2(1/probabilities)) for prob in range(1, probabilities+1)])
    return entropy

def jointEntropy(probabilities, newProbabilities):
    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))
    return jointEntropy


def plotConditionalEntropy(data, dt_string):
    plt.figure()
    for i in list(data.keys()):
        plt.plot(data[i][0], data[i][2])
    plt.xlabel('Moves')
    plt.ylabel('Conditional Entropy (bits)')
    plt.title('Entropy over Time in Chess')
    plt.savefig(f"figure-conditional-{dt_string}.png")
    plt.show()

def plotBestFitConditionalEntropy(data, dt_string):
    bestFitXs = []
    bestFitYs = []
    plt.figure()
    for i in list(data.keys()):
        plt.scatter(data[i][0], data[i][2])
        bestFitXs.append(data[i][0])
        bestFitYs.append(data[i][2])
    bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
    bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
    model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
    print(model)
    nModel = np.polyder(model)
    print(nModel)
    plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
    plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
    plt.legend(handlelength=3)
    plt.xlabel('Moves')
    plt.ylabel('Conditional Entropy  (bits)')
    plt.title('Conditional Entropy During the 2021 World Chess Championships', fontsize = 20)
    plt.savefig(f"figure-conditional-best-{dt_string}.png")
    plt.show()

def plotBestFitShannonEntropy(data, dt_string):
```

```
    bestFitXs = []
    bestFitYs = []
    plt.figure()
    for i in list(data.keys()):
        plt.scatter(data[i][0], data[i][1])
        bestFitXs.append(data[i][0])
        bestFitYs.append(data[i][1])
    bestFitXs = np.array(list(itertools.chain(*bestFitXs)))
    bestFitYs = np.array(list(itertools.chain(*bestFitYs)))
    model = np.poly1d(np.polyfit(bestFitXs, bestFitYs, 4))
    print(model)
    nModel = np.polyder(model)
    print(nModel)
    plt.plot(np.sort(bestFitXs), nModel(np.sort(bestFitXs)), linewidth=3.0, color="black", linestyle="dashdot", label="Derivative")
    plt.plot(np.sort(bestFitXs), model(np.sort(bestFitXs)), linewidth=3.0, color="black", label="Line of best fit")
    plt.legend(handlelength=3)
    plt.xlabel('Moves')
    plt.ylabel('Shannonian Entropy (bits)')
    plt.title('Shannon Entropy During the 2021 World Chess Championships',  fontsize = 20)
    plt.savefig(f"figure-shannon-best-{dt_string}.png")
    plt.show()

def plotShannonEntropy(data, dt_string):
    plt.figure()
    for i in list(data.keys()):
        plt.plot(data[i][0], data[i][1])
    plt.xlabel('Moves')
    plt.ylabel('Shannonian Entropy')
    plt.title('Entropy over Time in Chess')
    plt.savefig(f"figure-shannon-{dt_string}.png")
    plt.show()

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