



FINAL PROJECT REPORT

Geno Copertino



DECEMBER 9, 2022

Piece 1

Dataset Problems

The first piece of the final project mainly finding a dataset and creating and solving problems revolving around the sections we covered in the book. Firstly, this was fundamentally an interesting piece to tackle as this involved us using different formulas and distributions on practical real-world data we found. Personally, I tried to base my problems off two different datasets. I found data on different countries and their internet speeds according to Ookla and video game genre sales in the past 40 years by country. There were times it was tough to directly correlate a problem to my dataset, so I would have to get creative. I sometimes used other statistics I could find on the Internet to help build the problems. This piece easily took the 2nd most amount of time because we had to create and solve problems on content that took a whole semester to learn. There were times where I'd quickly have to look up a certain formula from an older section to make sure I could solve the problem. But, regardless of what my grades have been I genuinely do feel more proficient in this subject after this portion of the project.

Piece 1 also consisted of updating our formula sheet, writing this report, and updating our stats library programs as much as possible. I updated the formula sheet up to the independence density function (5.4). It was great learning in class how to cleanly make formulas in Microsoft Word so they look good. This was also a great piece to assign in general so that we always had a list of formulas that could be on the exam with us, being updated dynamically as we go. In terms of updating our stats formulas it seemed as if that part was more optional than the rest because of the "if possible" in the details for that portion. I wanted to go back and code a bunch more from the newer chapters but with all of these other pieces to complete, there wasn't enough time sadly.

In terms of this final report itself I didn't want to bloat it too much. I could copy word for word what I did in every piece and copy the content from the write-up's but I think those write-ups will speak for themselves. I'm trying to use this final report as a way to summarize the project briefly as a whole. Also, to give external opinions that wouldn't make sense to put on those reports will be on here.

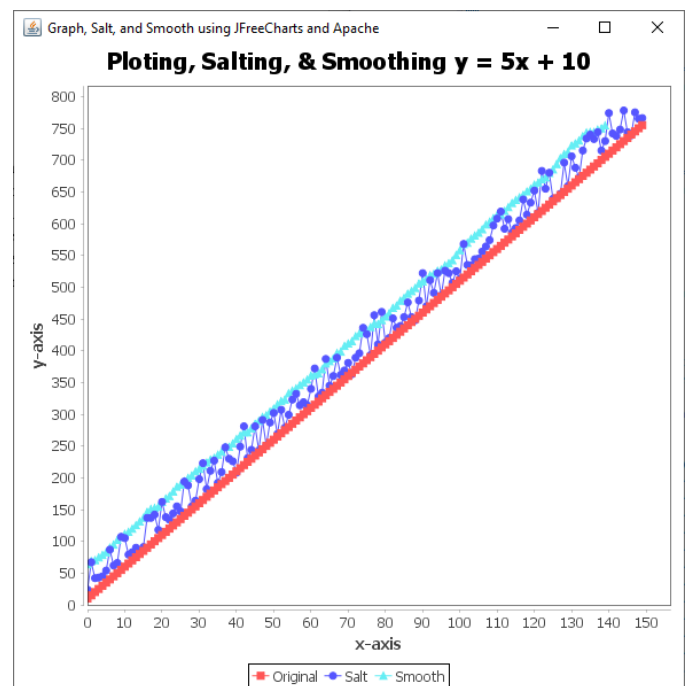
Piece 2

JFreeCharts and Apache

This was another interesting piece. It started off tricky since importing other libraries isn't intuitive or straightforward in Java. However, the fact that it can be done at all is amazing. It allows people to essentially make blueprints and do the bulk of the coding to make a program that would take tons of hours to make just take a couple or perform certain tasks entirely that wouldn't be possible with standard Java. Such was the case with JFreeCharts and Apache. Essentially, JFreeCharts is a chart library developed by David Gilbert that makes it easy for programmers to display professional quality charts in their application. The Apache Commons Mathematics library is a full-featured package of utility classes aimed at extending the functionality of the Java API.

Importing the libraries was done by right clicking on the folder your project was in (in Eclipse) and clicking "Properties". You then went Java Build Path on the left and clicked Libraries at the top. You then add all of the JAR files you downloaded. We needed to use JFreeCharts to help us code the graphing portion of this piece and Apache to code the smoothing portion. I believe I used JFreeCharts before on some other coding assignment in a past semester, but this was mostly relearning for me. First, you imported anything you needed from the libraries. Next, you created the plot itself, named the GUI, and initialized the size. Then, you created the panel, labeled the graph, and set the colors. Now, we have the groundwork laid to actually graph our function (original). We create two arrays, and two for loops to add all of the values needed. There were a lot of terms to figure out but eventually I created a graph of $y = 9x + 2$ (shown).

Salting the data I actually struggled with coding. The logic makes complete sense but I didn't submit a salter for project 1 so I still wasn't familiar with how coding it worked. I had to get some help on implementing that exact portion but in the end it made sense.



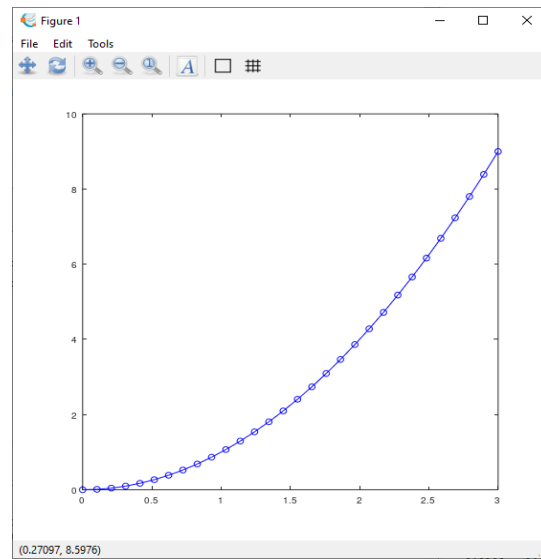
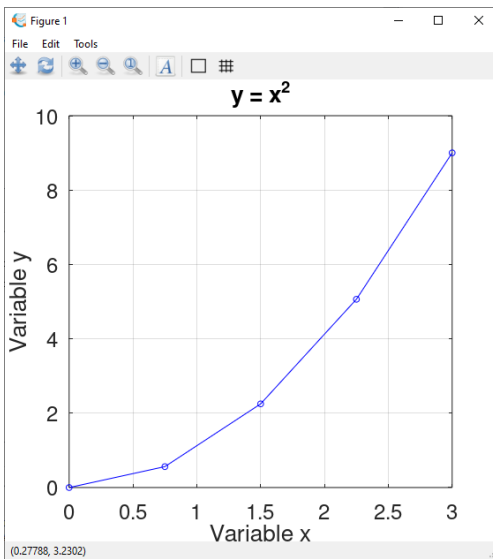
In conclusion, this piece really helped me understand a few things. One, how useful and important it is to know how to import other libraries. There seem to be quite a few free open-source libraries that can do amazing things so its important to understand how they work and how to implement them. JFreeCharts specifically seems to be one of the most popular ones for a

reason as it makes creating graphs very simple. Two, I improved my skill and understanding of JFreeCharts as a whole, which is an invaluable tool to have.

Piece 3

Octave

Piece 3 was easily my favorite piece of the project. It didn't take too long, and it was very fun learning the basics of a new little language. I had watched a tutorial on Octave that was around 1 hour 30 minutes long and it taught me just about everything I needed to know to complete the piece. In the report on what I learned about in the tutorial I talked about how the syntax of the code worked and how to declare variables. I also talked about how the workspace functioned. I started off with plotting the function $y = x^2$ normally which I learned from the tutorial. I began with a low number of points on the graph intentionally to showcase why the smoothing process is valuable. There were obvious kinks in my initial graph, but still that was the first portion of the assignment done. The tutorial talked about smoothing and so I went back and added a bunch more points and in my report I showed the new graph. It was logical how much better the new graph was at showing a trend or function more clearly. Here are the graphs side by side.



Piece 4

Normal, Gamma, and Beta Distributions

This portion of the final project had to do with researching and writing about three of the distributions from chapter 4 we didn't get to cover in class. They were the Normal, Gamma, and Beta distributions. Originally, this was one of the first pieces of the project I completed just because it was extremely simple. The distribution formulas all seemed scary at first. After looking at a couple of examples of each, I saw that they have a similar difficulty floor and ceiling like most of the other distributions we covered. Some of the symbols used in the formulas made them look especially complex.

When I first made my two page document on these distributions I felt that I included too many pictures and not enough actual effort in writing about the distributions. So, I went back and used the built in "equation" feature of word we learned about and wrote the formulas that way to take up less space. This allowed me to delve a little deeper into what the distributions are truly about. I still included graphs as sometimes a visual representation is needed to show what you are actually looking for.

In the future I think it is an amazing idea to assign graded write-ups (like this piece) on some content that you may not have the chance to get to in class. It provides a relatively easy grade for the students whilst still giving them a chance to cover more of the book. Overall, I had fun learning about these new distributions. I don't wanna delve too much into how they work here as the writeup I did on them details them extensively/

Piece 5

Poker Project

This was easily the hardest piece assigned and I still didn't get to finish the whole thing and I definitely needed help with how much I did get. First, we had to create a card class in which our card constructor takes in a rank and a suit since a card in a real deck has both of those attributes. We also need to create methods to check the rank and suit of a given card. Since, comparing strings or characters can be a process, the face cards were assigned numbers 10-14 (Ten – Ace). The suits were also assigned number values 1-4.

```
public class Card {  
    private char rank;  
    private char suit;  
  
    public Card(char inputRank, char inputSuit) {  
        rank = inputRank;  
        suit = inputSuit;  
    }  
  
    public char getRank() {  
        return this.rank;  
    }  
  
    public char getSuit() {  
        return this.suit;  
    }  
  
    //checks cards rank  
    public int checkRank() {  
        if (2 <= Character.getNumericValue(rank) && Character.getNumericValue(rank) <= 9) {  
            return Character.getNumericValue(rank);  
        }  
        int newRank = -1;  
        if (rank == 'T') {  
            newRank = 10;  
        }  
        if (rank == 'J') {  
            newRank = 11;  
        }  
        if (rank == 'Q') {  
            newRank = 12;  
        }  
        if (rank == 'K') {  
            newRank = 13;  
        }  
        if (rank == 'A') {  
            newRank = 14;  
        }  
        return newRank;  
    }  
  
    //checks card's suit  
    public int checkSuit() {  
        int newSuit = -1;  
        if (suit == 'H') {  
            newSuit = 1;  
        }  
        if (suit == 'D') {  
            newSuit = 2;  
        }  
        if (suit == 'S') {  
            newSuit = 3;  
        }  
        if (suit == 'C') {  
            newSuit = 4;  
        }  
        return newSuit;  
    }  
}
```

Next, we had to write the Deck class. The deck constructor creates a new deck arraylist and uses the generateCards() method that I'll talk about next. The generateCards() method made an arraylist of cards and used a nested for loop to create a 52 card hand deck. It loops 4 times total (each kind of suit) and 13 times (each kind of rank) and puts all of those cards in the deck. To draw a card from your deck, you check if the deck is empty first. If it isn't, a card will be removed from the deck and be ready to be drawn (future method will handle this). The drawHand() method is what takes those removed cards and adds them to your hand (five times as five cards are in a hand).

```
import java.util.ArrayList;
import java.util.Random;

public class Deck {

    private ArrayList<Card> deck;

    public Deck() {
        deck = new ArrayList<>();
        generateCards();
    }

    //generates cards into deck
    public void generateCards() {
        ArrayList<Card> cards = new ArrayList<Card>();
        String temp = "HDSC";
        String temp2 = "23456789TJQKA";

        for (int j = 0; j < 4; j++)
            for (int i = 0; i < 13; i++)
                cards.add(new Card(temp2.charAt(i), temp.charAt(j)));

        deck = cards;
    }

    //removes cards from deck and ready to be drawn
    public Card draw() {
        if(deck.isEmpty()) {
            return null;
        }

        Random random = new Random();
        int random1 = random.nextInt(deck.size());

        return deck.remove(random1);
    }

    //puts earlier removed cards into hand
    public ArrayList<Card> drawHand() {
        ArrayList<Card> hand = new ArrayList<>();
        for(int i = 0; i < 5; i++) {
            hand.add(draw());
        }
        return hand;
    }

    @Override
    public String toString() {
        String filler = "";
        for (Card test1 : deck) {
            filler += test1 + " ";
        }
        return filler;
    }
}
```

Finally, we have the HandCheck class. I was struggling a lot with the logic of how this worked, especially since we were working constantly with ArrayLists so I needed help to get the ball rolling. This class was responsible for checking for every kind of result you can have in a poker hand. Checking for a pair involved a lot of loops to compare each card to one another to check if there is a match. Once you have determined that there is only one pair through the if loops you change the frequency to 2. (Collections.frequency is something I forgot entirely about at first but

it turned out to be integral to the code was structured. The rest of the code extends off of the

```
//checks for four of a kind
public boolean isFour(ArrayList<Card> hand) {
    ArrayList<Integer> tempList = sorted(hand);
    if (Collections.frequency(tempList, tempList.get(2)) == 4) {
        return true;
    }

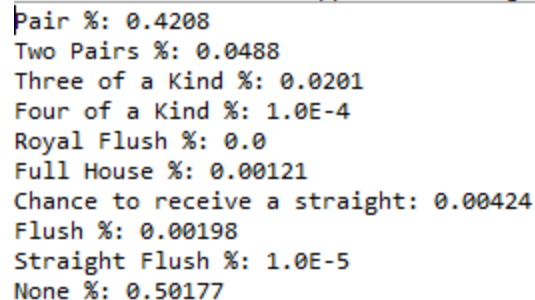
    return false;
}

//checks for full house
public boolean fullHouse(ArrayList<Card> hand) {
    ArrayList<Integer> tempList = sorted(hand);

    if (Collections.frequency(tempList, tempList.get(2)) != 3) {
        return false;
    }
    if (Collections.frequency(tempList, tempList.get(0)) == 2) {
        return true;
    }
    if (Collections.frequency(tempList, tempList.get(4)) == 2) {
        return true;
    }

    return false;
}
```

logic for that method. Here is an example of checking for four of a kind and full house. Eventually, after checking for every kind of test case (including none) we each for loop that uses the methods created to check for certain things 10000 times and divide whatever the count of the test case is by 10000 and get our probability of each event occurring. Here is a screenshot of the output.



```
Pair %: 0.4208
Two Pairs %: 0.0488
Three of a Kind %: 0.0201
Four of a Kind %: 1.0E-4
Royal Flush %: 0.0
Full House %: 0.00121
Chance to receive a straight: 0.00424
Flush %: 0.00198
Straight Flush %: 1.0E-5
None %: 0.50177
```

I was not able to make it playable (part 2). I started to implement some of the requirements but due to time constraints ultimately wasn't able to. Overall, I would say this project proved I need to make more of a habit coding in Java frequently as it shouldn't have been as hard as it was. Having mostly coded in Python the past two years, switching back over to an object-oriented language is extremely tough syntactially.