Hypothesis/Research Question:

This week's research questions and hypotheses I would like to answer and study are the following:

How does the size of the dataset affect the encryption time and the computing memory and RAM usage?

How do different parameters for the BFV and CKKS encryption schemes affect the encryption time and computing memory/RAM usage?

Process:

To answer this week's questions, I created mock datasets of varying sizes and encrypted them using both the BFV and CKKS schemes. I measured memory usage before and after encryption, as well as the time taken for the encryption process. First, I developed a function to generate datasets with 10 fixed features (columns) and a scalable number of rows. I tested datasets with 1,000, 10,000, and 100,000 rows, and I also varied the row count from 10,000 to 100,000 in increments of 10,000 to support the creation of a line graph illustrating scalability. The way I measured memory was by analyzing the activity monitor on my laptop and identifying which process it was, then I wrote down how much memory it took after the Python snippet was executed.

In addition to dataset size, I explored the impact of the polynomial modulus degree parameter on performance for both schemes, testing values of 2048, 4096, and 8192.The polynomial modulus degree controls the size of the underlying ciphertext and impacts both the noise budget and security level, which in turn affects encryption time and memory overhead, so analyzing it alongside dataset size allows for a more complete picture of the trade-offs involved.

Results:

Table 1: Encryption Time at Key Dataset Sizes (10 columns)

		1000 rows	**10000 rows**	**100000 rows**
BFV	**2048, [20,20]**	0.36 s	3.66 s	36.16 s
	4096, [30,20,30]	0.94 s	9.43 s	94.86 s
	8192, [30,20,30]	1.84 s	18.48 s	189.63 s
CKKS (global	**2048, [20,20]**	0.39 s	3.87 s	38.59 s

scale = 210)**	**4096, [30,20,30]**	1.03 s	10.20 s	103.24 s
	8192, [30,20,30]	2.06 s	20.66 s	207.99 s

Table 2: Memory Overhead at Key Dataset Sizes

		1000 rows	**10000 rows**	**100000 rows**
BFV	**2048, [20,20]**	262.1 MB	841.4 MB	6.44 GB
	4096, [30,20,30]	462.7 MB	2.67 GB	24.78 GB
	8192, [30, 20, 30]	740.5 MB	5.14 GB	49.22 GB
CKKS (global scale = 210)**	**2048, [20,20]**	263.0 MB	841.7 MB	6.44 GB
	4096, [30,20,30]	465.7 MB	2.67 GB	24.78 GB
	8192, [30, 20, 30]	741.0 MB	5.13 GB	49.23 GB

Table 3: Encryption Time for Scalability Analysis

		10000	**20000**	**30000**	**40000**	**50000**	**60000**	**70000**	**80000**	**90000**	**100000**
BFV	**2048, [20,20]**	3.61 s	7.27 s	11.11 s	14.79 s	18.40 s	22.22 s	26.31 s	30.00 s	33.84 s	37.46 s
	4096, [30,20,30]	9.89 s	19.72 s	29.70 s	38.91 s	49.29 s	57.85 s	66.44 s	75.79 s	85.88 s	95.64 s
	8192, [30,20,30]	18.41 s	37.05 s	56.19 s	75.56 s	96.38 s	116.34 s	136.80 s	154.80 s	173.99 s	194.32 s
CKKS (global scale = 210)**	**2048, [20,20]**	3.99 s	7.74 s	11.68 s	15.64 s	19.64 s	23.63 s	27.75 s	31.85 s	35.52 s	39.63 s
	4096, [30,20,30]	10.39 s	20.97 s	31.71 s	40.66 s	50.82 s	61.15 s	71.71 s	82.20 s	91.99 s	102.83 s
	8192, [30,20,30]	19.91 s	40.81 s	61.59 s	79.78 s	99.41 s	119.70 s	140.26 s	161.47 s	181.90 s	203.83 s

Table 4: Memory Overhead at Scalability Analysis

		10000	20000	30000	40000	50000	60000	70000	80000	90000	100000
BFV	**2048, [20,20]**	844.9 MB	1.44 GB	2.07 GB	2.70 GB	3.32 GB	3.94 GB	4.58 GB	5.20 GB	5.83 GB	6.45 GB
	4096, [30,20,30]	2.67 GB	5.12 GB	7.58 GB	10.04 GB	12.50 GB	14.95 GB	17.41 GB	19.87 GB	22.32 GB	24.78 GB
	8192, [30,20,30]	5.13 GB	10.03 GB	14.93 GB	19.83 GB	24.74 GB	29.64 GB	34.53 GB	39.43 GB	44.33 GB	49.23 GB
CKKS (global scale = 210)**	**2048, [20,20]**	835.9 MB	1.45 GB	2.07 GB	2.69 GB	3.32 GB	3.95 GB	4.56 GB	5.19 GB	5.81 GB	6.45 GB
	4096, [30,20,30]	2.67 GB	5.12 GB	7.58 GB	10.04 GB	12.50 GB	14.95 GB	17.41 GB	19.87 GB	22.33 GB	24.78 GB
	8192, [30,20,30]	5.13 GB	10.04 GB	14.93 GB	19.83 GB	24.74 GB	29.63 GB	34.53 GB	39.43 GB	44.32 GB	49.22 GB

Other Notes:

Given the linear growth pattern of the data and the serialized nature of the current implementation, the encryption process presents a clear opportunity for parallelization. Leveraging parallel computing during encryption could significantly improve performance by reducing runtime and increasing efficiency. This optimization would be especially impactful when working with large-scale datasets, making it a valuable consideration for future enhancements.