**LAB NOTEBOOK**

**INTERNSHIP RESEARCH**

**Aakash Singh**

**as.23bt8001@nitdgp.ac.in**

**Beginning 26 May 2025**

***Monday 26 May 2025***

1. **Created Github Repository**

I created a github repository to store all the

updates of internship covered till date.

1. **Gaining knowledge about further left chapters**

I am making a new pdf which covers topics of chapter 2 , so I can gain knowledge upon the topic and make a video on it as well.It will help me to grasp the knowledge in depth about the topic.

***Tuesday 27 May 2025***

1. **Worked upon the chapter**

Done the right work to understand the concept further.

1. **Helped Phd student a bit**

helped in finding and arranging some files (Diet information of mammals).

***Wednesday 28 May 2025***

1. **learned the basic of R programming**

Understood how code runs in R studio , Ran the code given to me but it showed error.

1. **completed chapter 2 full**

Now, Started chapter 3 in full pace

**Thursday 29 May 2025**

1. **Chapter 3 completed half**

Looked through all the task and ran all commands in compiler

1. **Learned some basic techniques of gene loss**

Bhaiya helped me to understand this in brief

**Friday 30 May 2025**

1. **Completed chapter 3 full**

Briefly completed chapter 3 along with making notes

**Monday 2 June 2025**

**Half completed chapter 4**

done the tasks present in book in the compiler and did the theory work

**Tuesday 3 June 2025**

**Chapter 4 completed full**

Done chapter 4 completely and heading forward to chapter 5

**Learned basics of Crispr**

Learned basics of crispr through youtube

**Wednesday 4 June 2025**

**Completed chapter 5 half**

Completed chapter 5 along with all task present in book

**Thursday 5 June 2025**

**Completed chapter 5**

Along with notes Completed chapter 5

**Uploaded video of Chapter 1 -4**

**Friday 6 June 2025**

**Half done Chapter 6**

Half completed chapter 6 along with some questions given in the book

**Monday 9 June 2025**

**Completed chapter 6 along with chapter 7 halfway**

Completed chapter 6 and basics of chapter 7 started

**Tuesday 10 June 2025**

**Completed chapter 7 and Chapter 8**

Started off with chapter 9 and learned things about debugging

**Wednesday 11 June 2025**

Completed half reading chapter 9 and made some notes

**Thursday 12 June 2025**

Completed with chapter 9 and done all task in it and made notes

**Friday 13 June 2025**

Started off with chapter 10 and learned the basics of blasting and its functions.

**Monday 16 June 2025**

Completed with chapter 10 and done all the tasks in the chapters.

**Tuesday 17 June 2025**

Started and completed Half chapter 11 and looking forward to doing some real work.

**Wednesday 18 June 2025**

Completed with chapter 11 along with all the tasks in it.

**Thursday 19 June 2025**

Started off with final chapter 12 and looking forward to completing it.

**Friday 20 June 2025**

Completed all the chapters and looking forward to making videos of all chapters as asked by Sir.

**Monday 23 June 2025**

Made chapter 1-2 videos each of 30 min as asked by Sir.

**Tuesday 24 June 2025**

Completed making Videos for chapter 5.

**Wednesday 25 June 2025**

Completed making Videos for chapter 6.

**Thursday 26 June 2025**

Completed making Videos for chapter 7.

**Friday 27 June 2025**

Completed making Videos for chapter 8.

**Monday 30 June 2025**

Completed making Videos for chapter 9-10.

**Tuesday 1 July 2025**

Completed making videos for chapter 11 and 12.

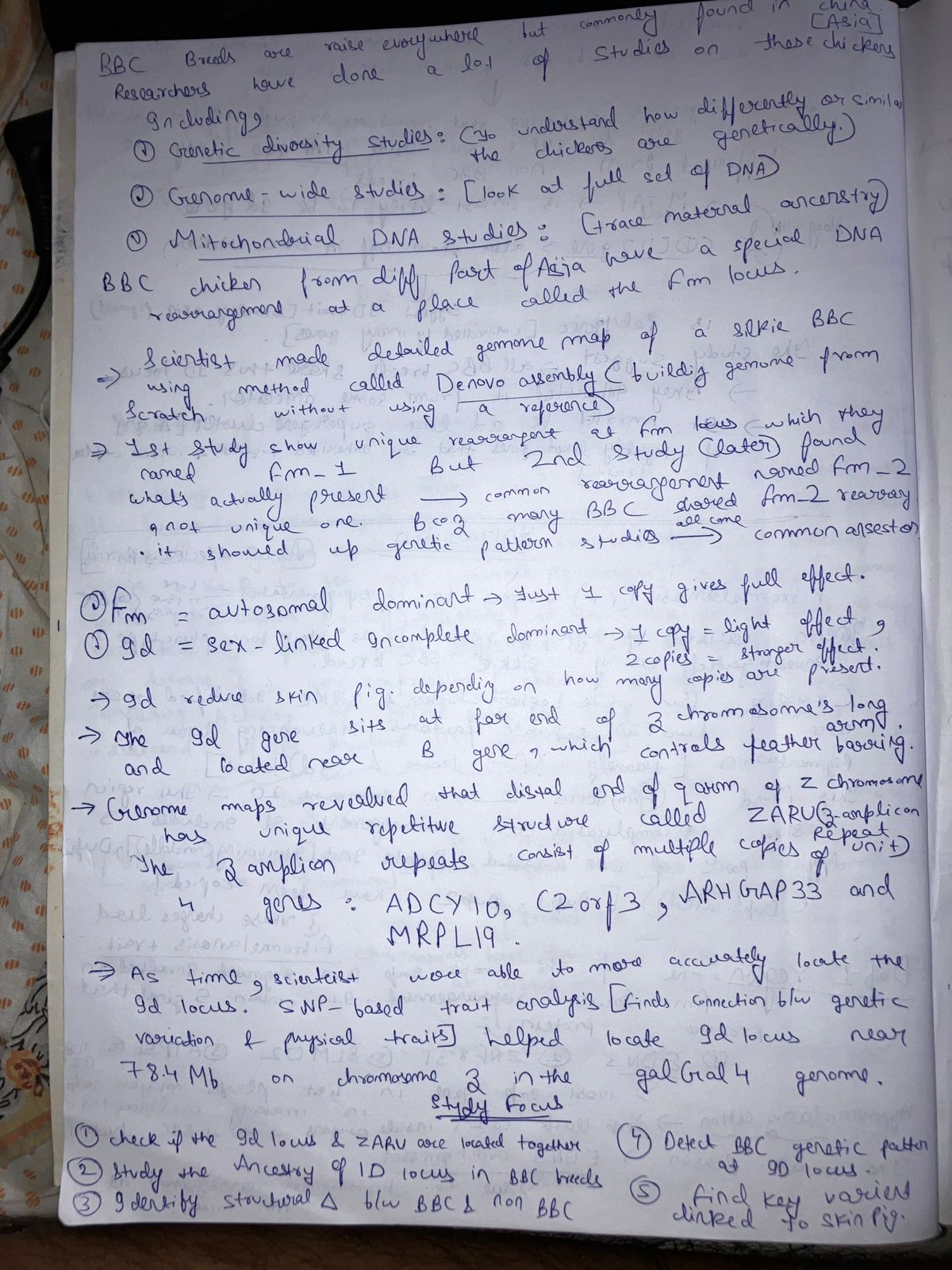
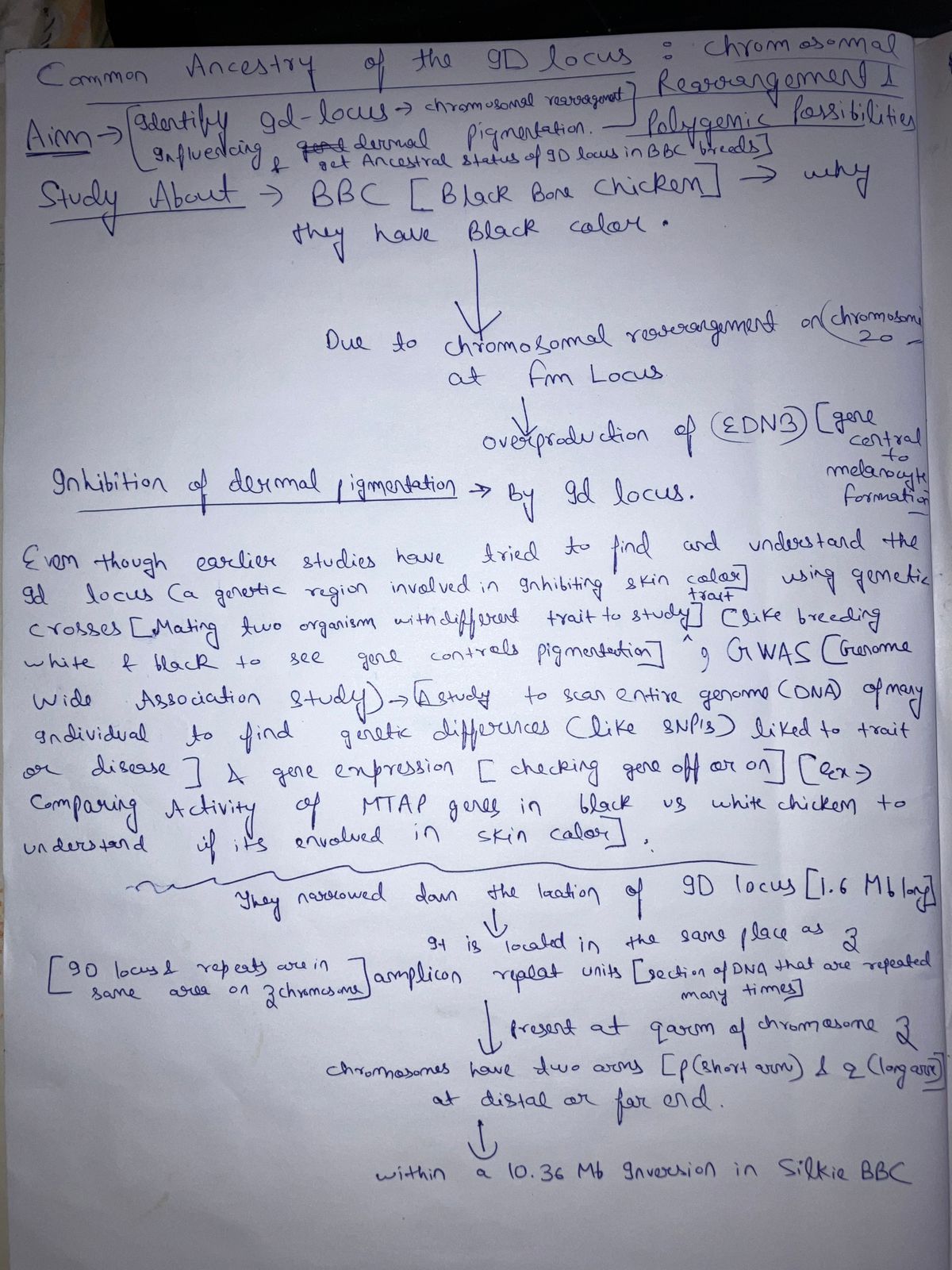
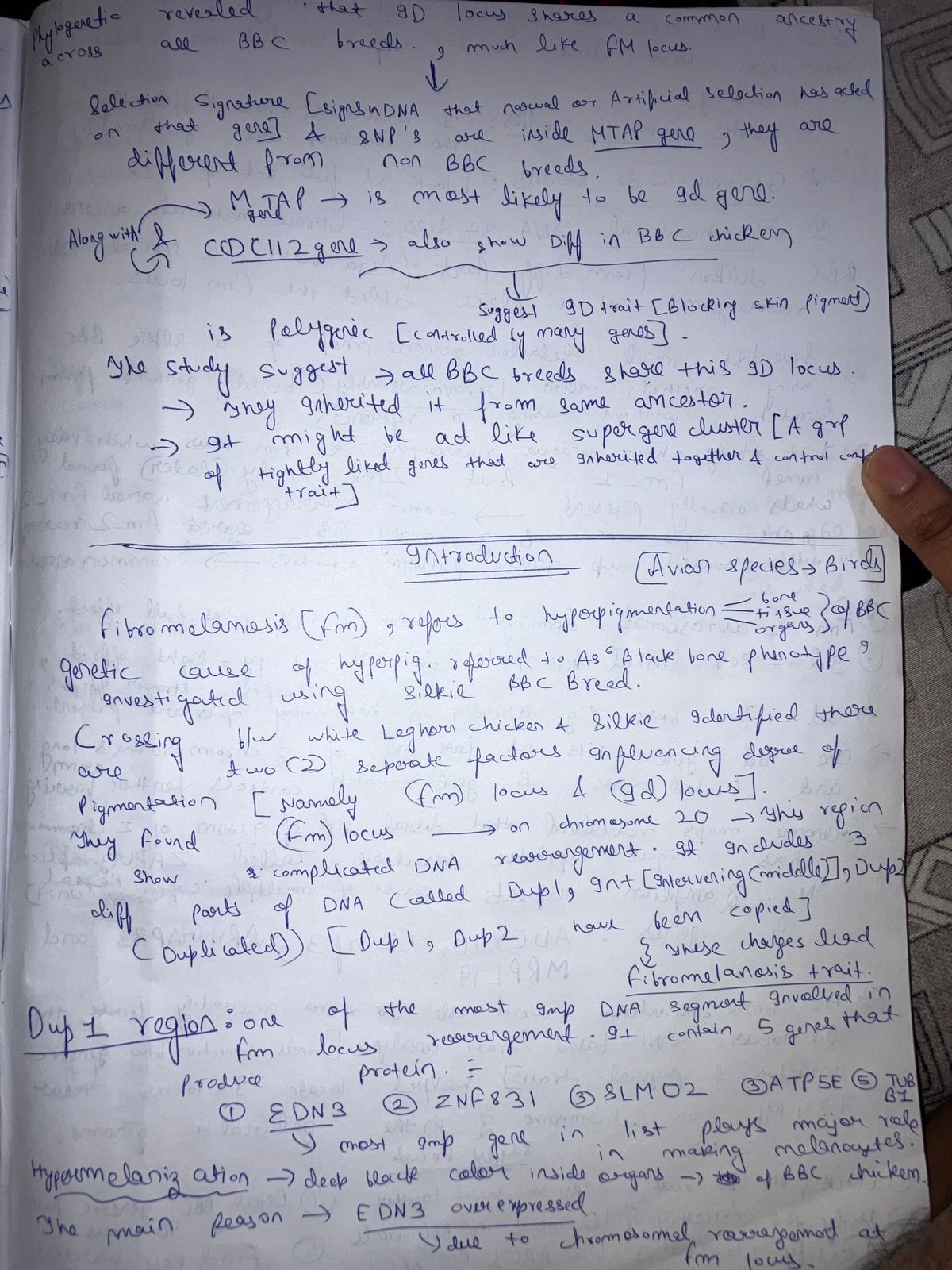
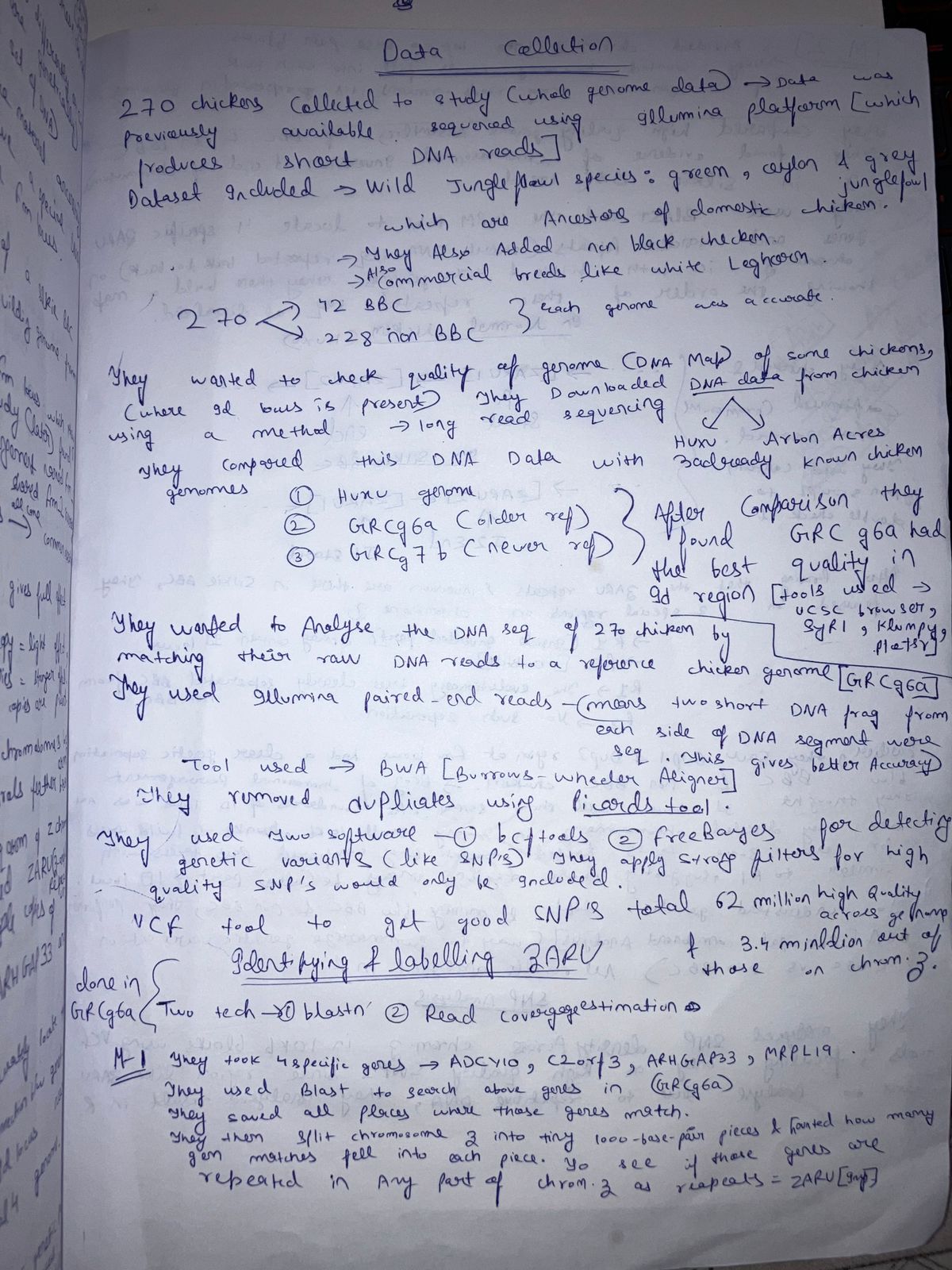
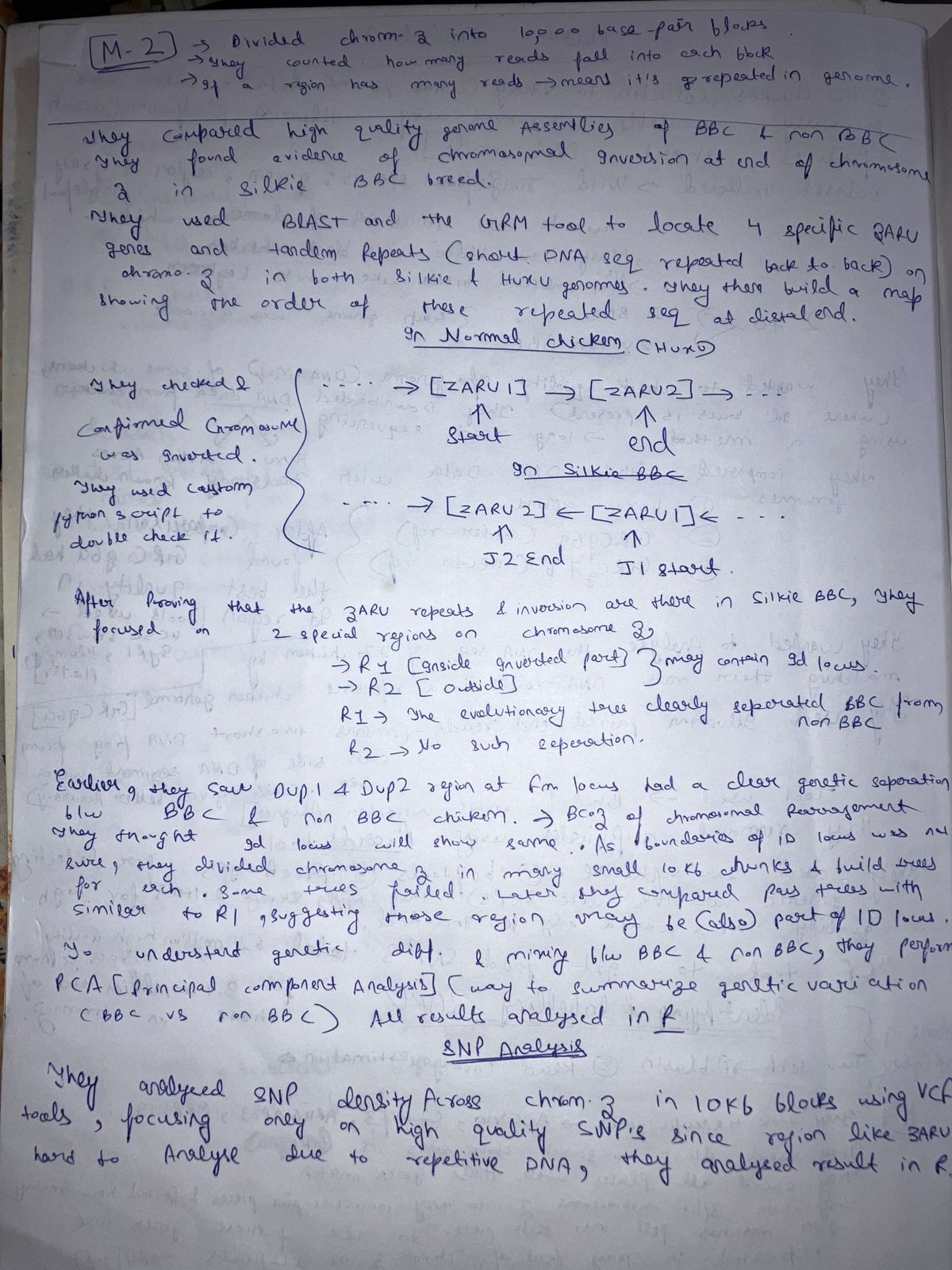
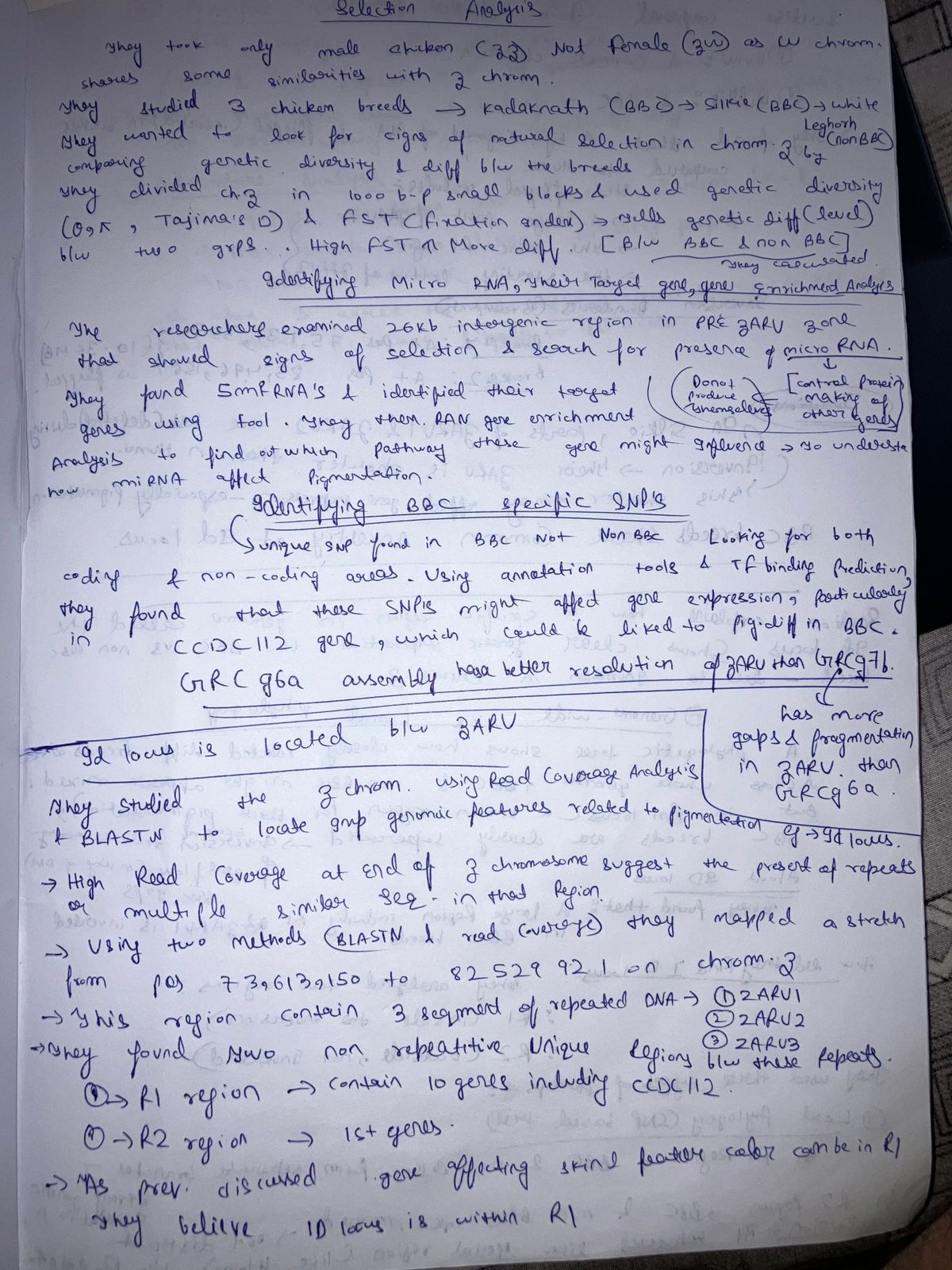
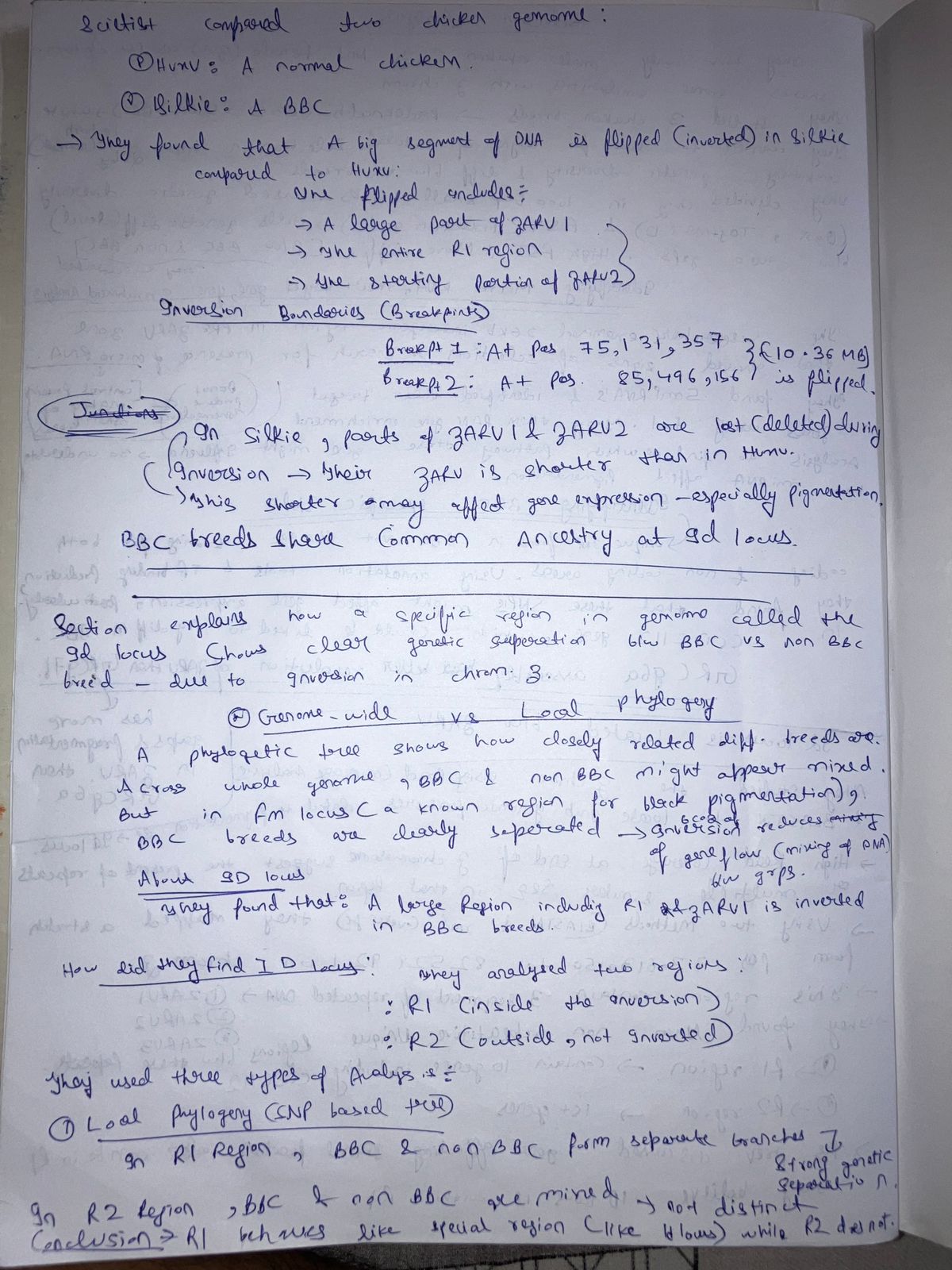
**Wednesday 2 July 2025**

I started reading a research paper given to me by sir . The research focuses on finding the specific location of genes responsible for Black color bones and skin in black bone chicken.

“https://link.springer.com/article/10.1007/s00239-025-10233-z”

**Thursday 3 July-Monday 7 July 2025**

Read the paper thoroughly and understand all the minor details about it along with making notes.

****

**Tuesday 8 July-Tuesday 15 July 2025**

Using knowledge of Coding with help of Internet created Simulation Script which not only create genomic bed files but also performs Mutations of the genes (example : Deletion, Duplication, Inversion , Insertion)

**15 July - 20 July 2025**

The code Present in Original Paper fails to analyze results for deletion and insertion of genes which I fixed up to an extent . All the files along with Readme File is present in repository :

“<https://github.com/Zuhn0/simulation/tree/main>”

These are the codes with their File names :

**Extract\_rows\_from\_bed.py**

**Code**:

import argparse

def extract\_rows(infile, outfile, start\_row, end\_row):

with open(infile, 'r') as fin:

lines = fin.readlines()

# BED file lines start from index 0

selected\_lines = lines[start\_row - 1:end\_row] # Adjust for 1-based input

with open(outfile, 'w') as fout:

fout.writelines(selected\_lines)

print(f"Extracted rows {start\_row} to {end\_row} into '{outfile}'.")

if \_\_name\_\_ == "\_\_main\_\_":

parser = argparse.ArgumentParser(description="Extract specific row range from a BED file.")

parser.add\_argument("infile", help="Input BED file")

parser.add\_argument("outfile", help="Output file for extracted rows")

parser.add\_argument("start", type=int, help="Start row number (1-based)")

parser.add\_argument("end", type=int, help="End row number (inclusive, 1-based)")

args = parser.parse\_args()

extract\_rows(args.infile, args.outfile, args.start, args.end)

**simulation.py**

**Code**:

import argparse

import random

import numpy as np

def simulate\_bed(num\_regions, mean\_len, sd\_len, gene\_entries, mean\_chrom\_gap, sd\_chrom\_gap, output):

chroms = [f"chr{i}" for i in range(1, 2006)]

total\_chroms = len(chroms)

# Parse gene list and weights

if len(gene\_entries) % 2 != 0:

raise ValueError("Each gene must be followed by its percentage.")

genes = gene\_entries[::2]

weights = list(map(float, gene\_entries[1::2]))

if sum(weights) != 100:

raise ValueError("Gene percentages must sum to 100.")

regions\_per\_chrom = [num\_regions // total\_chroms] \* total\_chroms

for i in range(num\_regions % total\_chroms):

regions\_per\_chrom[i] += 1

regions = []

global\_position = 0

for chrom\_index, chrom in enumerate(chroms):

num\_regions\_this\_chrom = regions\_per\_chrom[chrom\_index]

if chrom\_index > 0:

gap = 0

while gap < 1:

gap = int(round(np.random.normal(mean\_chrom\_gap, sd\_chrom\_gap)))

global\_position += gap

current\_pos = global\_position

for \_ in range(num\_regions\_this\_chrom):

length = 0

while length < 1:

length = int(round(np.random.normal(mean\_len, sd\_len)))

start = current\_pos

end = start + length

gene = random.choices(genes, weights=weights, k=1)[0]

regions.append((chrom, start, end, length, gene))

current\_pos = end + 1

global\_position = current\_pos

chrom\_index\_map = {f"chr{i}": i for i in range(1, 2006)}

regions.sort(key=lambda x: (chrom\_index\_map.get(x[0], 0), x[1]))

with open(output, 'w') as f:

for chrom, start, end, length, gene in regions:

f.write(f"{chrom}\t{start}\t{end}\t{length}\t{gene}\n")

if \_\_name\_\_ == "\_\_main\_\_":

parser = argparse.ArgumentParser(description="Simulate BED file with gene type percentages")

parser.add\_argument("-n", "--number", type=int, required=True, help="Total number of regions")

parser.add\_argument("-L", "--mean\_length", type=float, required=True, help="Mean region length")

parser.add\_argument("-S", "--sd\_length", type=float, required=True, help="Standard deviation of region length")

parser.add\_argument("-g", "--genes", nargs="+", required=True,

help="List of gene types and their percentages. Example: -g ADCY10 40 MRPL19 30 C2orf3 30")

parser.add\_argument("-D", "--mean\_chrom\_gap", type=float, required=True, help="Mean distance between chromosomes")

parser.add\_argument("-T", "--sd\_chrom\_gap", type=float, required=True, help="Standard deviation of chromosome gap")

parser.add\_argument("-o", "--output", default="simulated\_output.bed", help="Output BED file name")

args = parser.parse\_args()

simulate\_bed(

args.number,

args.mean\_length,

args.sd\_length,

args.genes,

args.mean\_chrom\_gap,

args.sd\_chrom\_gap,

args.output

)

**mutationsimulation. py**

**Code:**

import argparse, sys

def parse\_args():

parser = argparse.ArgumentParser(description="Simulate mutations on a BED file")

parser.add\_argument("infile", help="Input BED filename (tab-delimited, 5 columns)")

parser.add\_argument("outfile", help="Output mutated BED filename")

parser.add\_argument("-d", "--delete", action="append",

help="Region to delete (format CHR:START-END or gene name)")

parser.add\_argument("--dup", action="append",

help="Region to duplicate (format CHR:START-END[@CHR:NEWSTART])")

parser.add\_argument("-v", "--invert", action="append",

help="Region to invert (format CHR:START-END or gene name)")

parser.add\_argument("-i", "--insert", action="append",

help="Insertion (format CHR:POS:LENGTH:GENE)")

return parser.parse\_args()

def main():

args = parse\_args()

try:

with open(args.infile) as f:

bed\_records = []

for line in f:

line = line.strip()

if not line: continue

cols = line.split("\t")

if len(cols) != 5:

print(f"Error: Invalid BED line (expected 5 columns): {line}")

sys.exit(1)

chrom, start, end, length, gene = cols

try:

start = int(start); end = int(end); length = int(length)

except ValueError:

print(f"Error: Non-integer value in BED line: {line}")

sys.exit(1)

bed\_records.append([chrom, start, end, length, gene])

except FileNotFoundError:

print(f"Error: Input file '{args.infile}' not found.")

sys.exit(1)

# Perform deletions

if args.delete:

for reg in args.delete:

if "@" in reg:

gene, start\_str = reg.split("@", 1)

try:

start = int(start\_str)

except ValueError:

print(f"Error: Invalid start coordinate in delete '{reg}'")

sys.exit(1)

matches = [rec for rec in bed\_records if rec[4] == gene and rec[1] == start]

else:

print(f"Error: For deletion, use format GENE@START (e.g. G1@502)")

sys.exit(1)

if not matches:

print(f"Error: Deletion target '{reg}' not found.")

sys.exit(1)

for rec in matches:

bed\_records.remove(rec)

# duplications

if args.dup:

for reg in args.dup:

parts = reg.split("@")

src = parts[0]

chrom, rest = src.split(":", 1)

start, end = map(int, rest.split("-", 1))

matches = [rec for rec in bed\_records if rec[0]==chrom and rec[1]==start and rec[2]==end]

if not matches:

print(f"Error: Duplication target '{src}' not found.")

sys.exit(1)

orig = matches[0]

new\_chrom = orig[0]

new\_start = orig[1]

new\_end = orig[2]

if len(parts) > 1 and parts[1]:

dest = parts[1]

if ":" in dest:

new\_chrom, new\_start = dest.split(":")

new\_start = int(new\_start)

else:

new\_start = int(dest)

length = orig[2] - orig[1]

new\_end = new\_start + length

else:

length = orig[2] - orig[1]

new\_end = new\_start + length

# Shift downstream genes

for rec in bed\_records:

if rec[0] == new\_chrom and rec[1] >= new\_start:

rec[1] += length

rec[2] += length

for rec in bed\_records:

if rec[0] > new\_chrom:

rec[1] += length

rec[2] += length

new\_length = new\_end - new\_start

bed\_records.append([new\_chrom, new\_start, new\_end, new\_length, orig[4]])

# Handle inversions

if args.invert:

for reg in args.invert:

if ":" in reg and "-" in reg:

chrom, rest = reg.split(":", 1)

start, end = map(int, rest.split("-", 1))

matches = [rec for rec in bed\_records if rec[0]==chrom and rec[1]==start and rec[2]==end]

else:

# Match ALL records by gene name

matches = [rec for rec in bed\_records if rec[4] == reg]

if not matches:

print(f"Error: Inversion target '{reg}' not found.")

sys.exit(1)

for rec in matches:

# Flip start and end literally

rec[1], rec[2] = rec[2], rec[1] # reversed coordinates

rec[3] = -(abs(rec[1] - rec[2])) # negative length

# Gene name stays the same

# Handle insertions

if args.insert:

for reg in args.insert:

parts = reg.split(":", 3)

if len(parts) != 4:

print(f"Error: Insert format must be CHR:POS:LENGTH:GENE, got '{reg}'")

sys.exit(1)

chrom, pos\_str, length\_str, gene = parts

pos = int(pos\_str)

length = int(length\_str)

overlap = [rec for rec in bed\_records if rec[0]==chrom and rec[1] < pos < rec[2]]

if overlap:

print(f"Error: Insertion at {chrom}:{pos} overlaps existing gene {overlap[0][4]}")

sys.exit(1)

for rec in bed\_records:

if rec[0]==chrom and rec[1] >= pos:

rec[1] += length

rec[2] += length

for rec in bed\_records:

if rec[0] > chrom:

rec[1] += length

rec[2] += length

new\_rec = [chrom, pos, pos+length, length, gene]

bed\_records.append(new\_rec)

# Sort output

def chr\_sort\_key(rec):

chrom = rec[0]

if chrom.startswith("chr"):

chrom = chrom[3:]

try:

return (int(chrom), rec[1])

except ValueError:

return (chrom, rec[1])

bed\_records.sort(key=chr\_sort\_key)

# Write to output file

try:

with open(args.outfile, 'w') as out:

for chrom, start, end, length, gene in bed\_records:

out.write(f"{chrom}\t{start}\t{end}\t{length}\t{gene}\n")

except IOError as e:

print(f"Error writing output file: {e}")

sys.exit(1)

print(f"Mutated BED written to {args.outfile}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Updated\_find\_pattern\_first\_p-value\_fixed.py**

**Code:**

import sys

import math

def read\_bed\_file(filename):

with open(filename, 'r') as f:

return [line.strip().split() for line in f if line.strip()]

def compute\_differences(data):

differences = []

for i in range(1, len(data)):

diff = int(data[i][1]) - int(data[i-1][2])

differences.append(diff)

return differences

def extract\_lengths(data):

return [int(row[3]) for row in data]

def extract\_genes(data):

return [row[4] for row in data]

def match\_pattern(query\_genes, query\_lengths, query\_diffs, segment, tol\_len, tol\_diff):

segment\_genes = [row[4] for row in segment]

segment\_lengths = extract\_lengths(segment)

segment\_diffs = compute\_differences(segment)

inserted = []

deleted = []

match\_len = match\_diff = match\_exact = 0

i = j = 0

while i < len(query\_genes) and j < len(segment\_genes):

if query\_genes[i] == segment\_genes[j]:

diff\_match = abs(query\_diffs[i-1] - segment\_diffs[j-1]) <= tol\_diff if i > 0 and j > 0 else True

len\_match = abs(query\_lengths[i] - segment\_lengths[j]) <= tol\_len

if diff\_match:

match\_diff += 1

if len\_match:

match\_len += 1

if len\_match and diff\_match:

match\_exact += 1

i += 1

j += 1

elif query\_genes[i] not in segment\_genes[j:j+2]:

deleted.append((i, query\_genes[i]))

i += 1

else:

inserted.append((j, segment\_genes[j]))

j += 1

while i < len(query\_genes):

deleted.append((i, query\_genes[i]))

i += 1

while j < len(segment\_genes):

inserted.append((j, segment\_genes[j]))

j += 1

p\_value = math.exp(-1 \* (match\_exact + 0.5 \* match\_len + 0.25 \* match\_diff))

return {

"match\_exact": match\_exact,

"match\_len": match\_len,

"match\_diff": match\_diff,

"p\_value": p\_value,

"inserted": inserted,

"deleted": deleted

}

def find\_patterns(bed\_data, gene\_list, len\_list, diff\_list, tol\_len, tol\_diff):

matches = []

n = len(gene\_list)

for i in range(len(bed\_data) - n + 1):

segment = bed\_data[i:i+n]

result = match\_pattern(gene\_list, len\_list, diff\_list, segment, tol\_len, tol\_diff)

matches.append((i, i+n-1, 'forward', result))

matches.sort(key=lambda x: x[3]['p\_value'])

return matches[:3]

def print\_result(start, end, direction, result):

print(f"{start}\t{end}\t{direction}\t"

f"{result['match\_exact']}\t"

f"{result['match\_len']}\t"

f"{result['match\_diff']}\t"

f"{result['p\_value']:.5f}\t"

f"{','.join(g for \_, g in result['inserted']) or '-'}\t"

f"{','.join(g for \_, g in result['deleted']) or '-'}")

def main():

if len(sys.argv) != 7:

print("Usage: python3 updated\_find\_pattern\_first\_p-value\_fixed.py <bed\_file> <gene\_list> <len\_list> <diff\_list> <tol\_len> <tol\_diff>")

sys.exit(1)

bed\_file = sys.argv[1]

gene\_list = sys.argv[2].split(",")

len\_list = list(map(int, sys.argv[3].split(",")))

diff\_list = list(map(int, sys.argv[4].split(",")))

tol\_len = int(sys.argv[5])

tol\_diff = int(sys.argv[6])

bed\_data = read\_bed\_file(bed\_file)

top\_matches = find\_patterns(bed\_data, gene\_list, len\_list, diff\_list, tol\_len, tol\_diff)

# Print header

print("Start\tEnd\tDirection\tExact\_Length\_Matches\tWithin\_Tolerance\_Length\_Matches\tWithin\_Tolerance\_Diff\_Matches\tP\_value\tInserted\_Genes\tDeleted\_Genes")

for start, end, direction, result in top\_matches:

print\_result(start, end, direction, result)

if \_\_name\_\_ == "\_\_main\_\_":

main()