<u>SUBMITTED BY</u>- *VIBHANSHU SINGH (M.TECH COMPUTATIONAL BIOLOGY)*

Task 1: Data Handling and Statistical Analysis

Objective: Assess candidates' ability to handle complex data and apply statistical methods effectively.

Background: CpG methylation is an epigenetic marker that varies across tissue types. However, the methylation status of a single CpG site is unreliable as a biomarker due to errors introduced by bisulfite sequencing, sampling techniques, and biological variability.

Definition: Phased Methylation Pattern (PMP) is a unique set of coordinates that includes the DNA strand ('f' for forward (+) or 'r' for reverse (-)), the relative positions of three CpG sites on the same strand (e.g., x:y:z), and their methylation status (e.g., '000' for all unmethylated or '111' for all methylated). It represents a combined epigenetic signature across these CpG sites.

Hypothesis: Phased methylation patterns (PMPs) can act as reliable biomarkers to differentiate tissue types, providing higher specificity compared to individual CpG sites.

• Strand: Indicates the DNA strand ('f' or 'r').
• CpG Coordinates: Relative positions of three CpG sites (x:y:z).
• Methylation Status: Eight possible patterns ('000' to '111').
• Sample ID: Unique identifier for each sample.
• Replicate: Indicates technical replicates.
• Tissue: Tissue type (Tissue #1 or Tissue #2).

Sub-tasks:

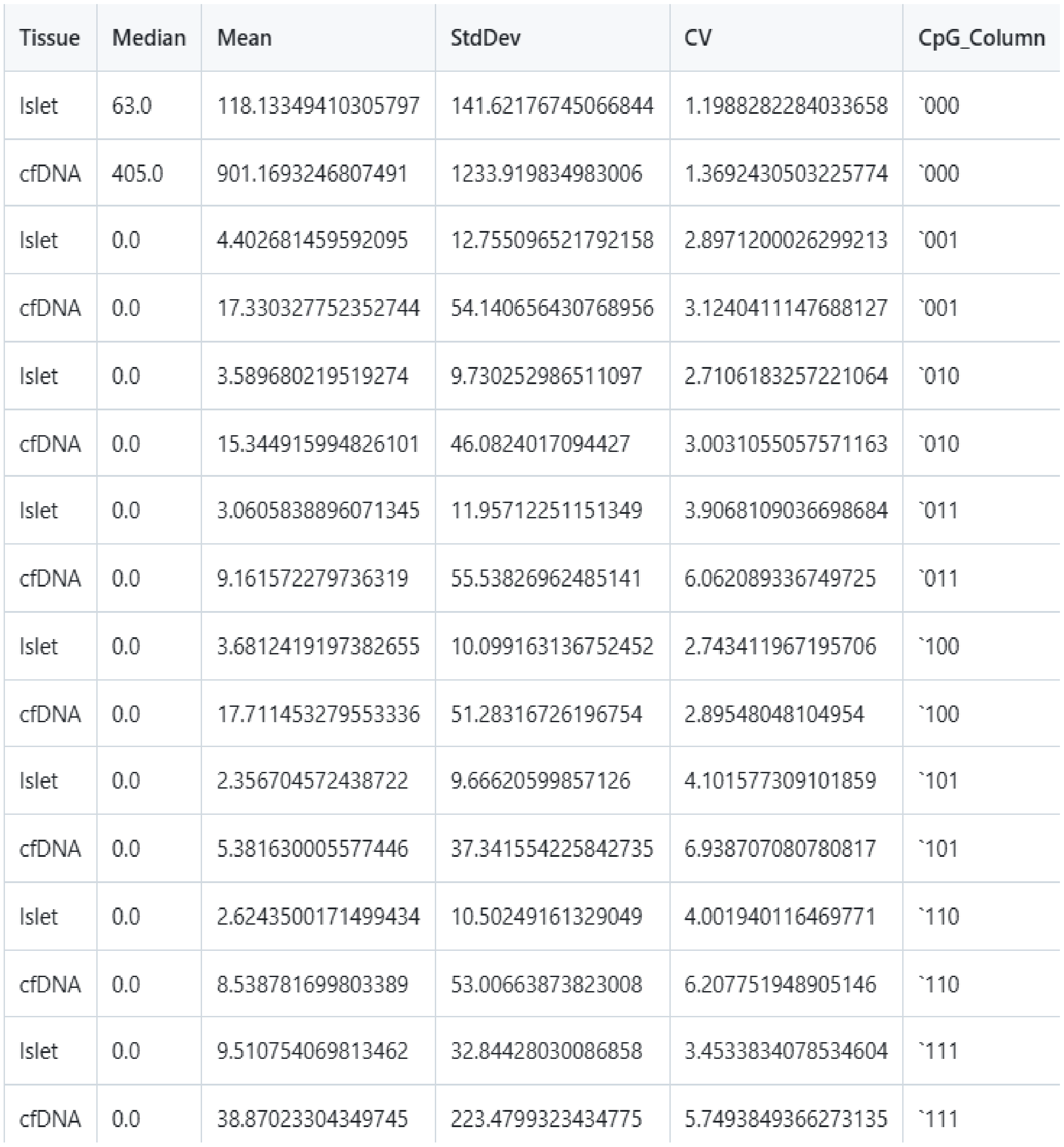
1. Coverage Analysis

a. Calculate the median and coefficient of variation (CV) for single CpG coverage in each tissue. coverage analysis ensures the integrity and reliability of methylation data by addressing sequencing biases, variability, and depth requirements. It is foundational to validating hypotheses like the use of PMPs as tissue-specific biomarkers.

Code-

```
import pandas as pd


df = pd.read_csv('PupilBioTest.csv')

# Columns for single CpG coverage
cpg_columns = ['`000', '`001', '`010', '`011', '`100', '`101', '`110', '`111']

# dictionary to store metrics for each cpg column
results = []

# loop through each CpG column
for cpg in cpg_columns:
    # grouped by tissue
    grouped = df.groupby('Tissue')[cpg].agg(
        Median='median',
        Mean='mean',
        StdDev='std'
    ).reset_index()


    grouped['CV'] = grouped['StdDev'] / grouped['Mean']

    # add a column to indicate which CpG column this analysis belongs to
    grouped['CpG_Column'] = cpg

    # append the result for this CpG column to the results list
    results.append(grouped)

# merge all results
final_results = pd.concat(results, ignore_index=True)

final_results.to_csv('single_cpg_coverage_statistics.csv', index=False)
print(final_results)
```

Result



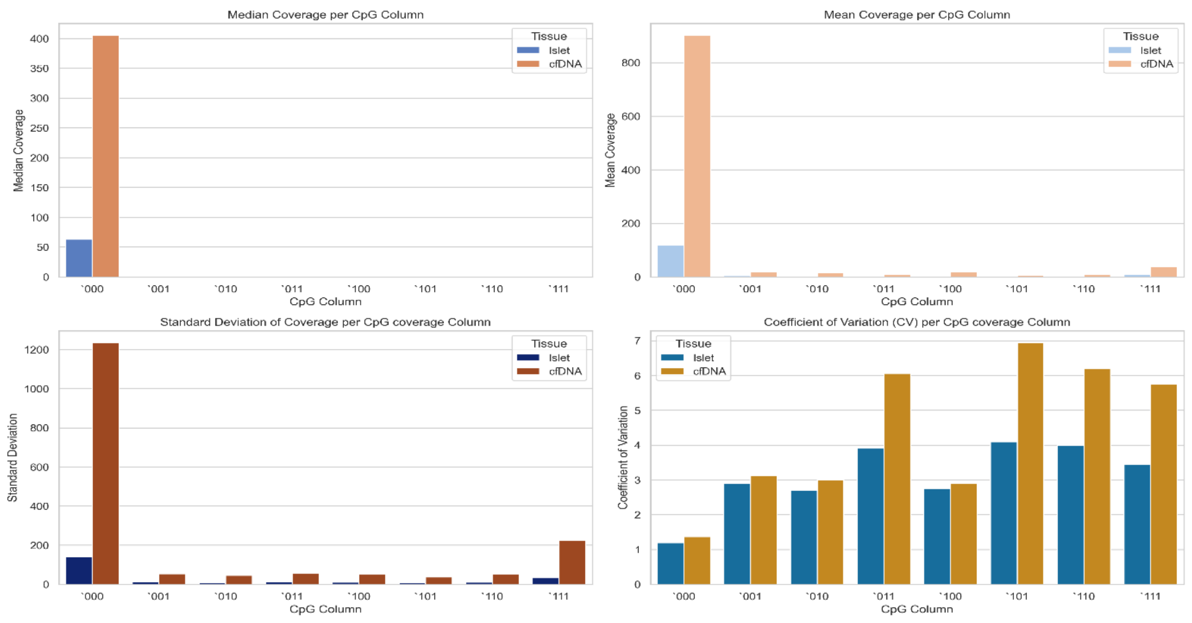
Tissue	Median	Mean	StdDev	CV	CpG_Column
Islet	63.0	118.13349410305797	141.62176745066844	1.1988282284033658	`000
cfDNA	405.0	901.1693246807491	1233.919834983006	1.3692430503225774	`000
Islet	0.0	4.402681459592095	12.755096521792158	2.8971200026299213	`001
cfDNA	0.0	17.330327752352744	54.140656430768956	3.1240411147688127	`001
Islet	0.0	3.589680219519274	9.730252986511097	2.7106183257221064	`010
cfDNA	0.0	15.344915994826101	46.0824017094427	3.0031055057571163	`010
Islet	0.0	3.0605838896071345	11.95712251151349	3.9068109036698684	`011
cfDNA	0.0	9.161572279736319	55.53826962485141	6.062089336749725	`011
Islet	0.0	3.6812419197382655	10.099163136752452	2.743411967195706	`100
cfDNA	0.0	17.711453279553336	51.28316726196754	2.89548048104954	`100
Islet	0.0	2.356704572438722	9.66620599857126	4.101577309101859	`101
cfDNA	0.0	5.381630005577446	37.341554225842735	6.938707080780817	`101
Islet	0.0	2.6243500171499434	10.50249161329049	4.001940116469771	`110
cfDNA	0.0	8.538781699803389	53.00663873823008	6.207751948905146	`110
Islet	0.0	9.510754069813462	32.84428030086858	3.4533834078534604	`111
cfDNA	0.0	38.87023304349745	223.4799323434775	5.7493849366273135	`111

b. Generate plots summarizing the coverage statistics.

Code-

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#plot
results = pd.read_csv('single_cpg_coverage_statistics.csv')


sns.set(style="whitegrid")
plt.figure(figsize=(16, 10))

# 1. Median plot
plt.subplot(2, 2, 1)
sns.barplot(data=results, x='CpG_Column', y='Median', hue='Tissue', palette='muted')
plt.title('Median Coverage per CpG Column')
plt.xlabel('CpG Column')
plt.ylabel('Median Coverage')

# 2. Mean plot
plt.subplot(2, 2, 2)
sns.barplot(data=results, x='CpG_Column', y='Mean', hue='Tissue', palette='pastel')
plt.title('Mean Coverage per CpG Column')
plt.xlabel('CpG Column')
plt.ylabel('Mean Coverage')

# 3. Standard Deviation plot
plt.subplot(2, 2, 3)
sns.barplot(data=results, x='CpG_Column', y='StdDev', hue='Tissue', palette='dark')
plt.title('Standard Deviation of Coverage per CpG coverage Column')
plt.xlabel('CpG Column')
plt.ylabel('Standard Deviation')

# 4. CV plot
plt.subplot(2, 2, 4)
sns.barplot(data=results, x='CpG_Column', y='CV', hue='Tissue', palette='colorblind')
plt.title('Coefficient of Variation (CV) per CpG coverage Column')
plt.xlabel('CpG Column')
plt.ylabel('Coefficient of Variation')

plt.tight_layout()
plt.savefig('coverage_statistics_plots.png', dpi=300)
plt.show()
```

Result –

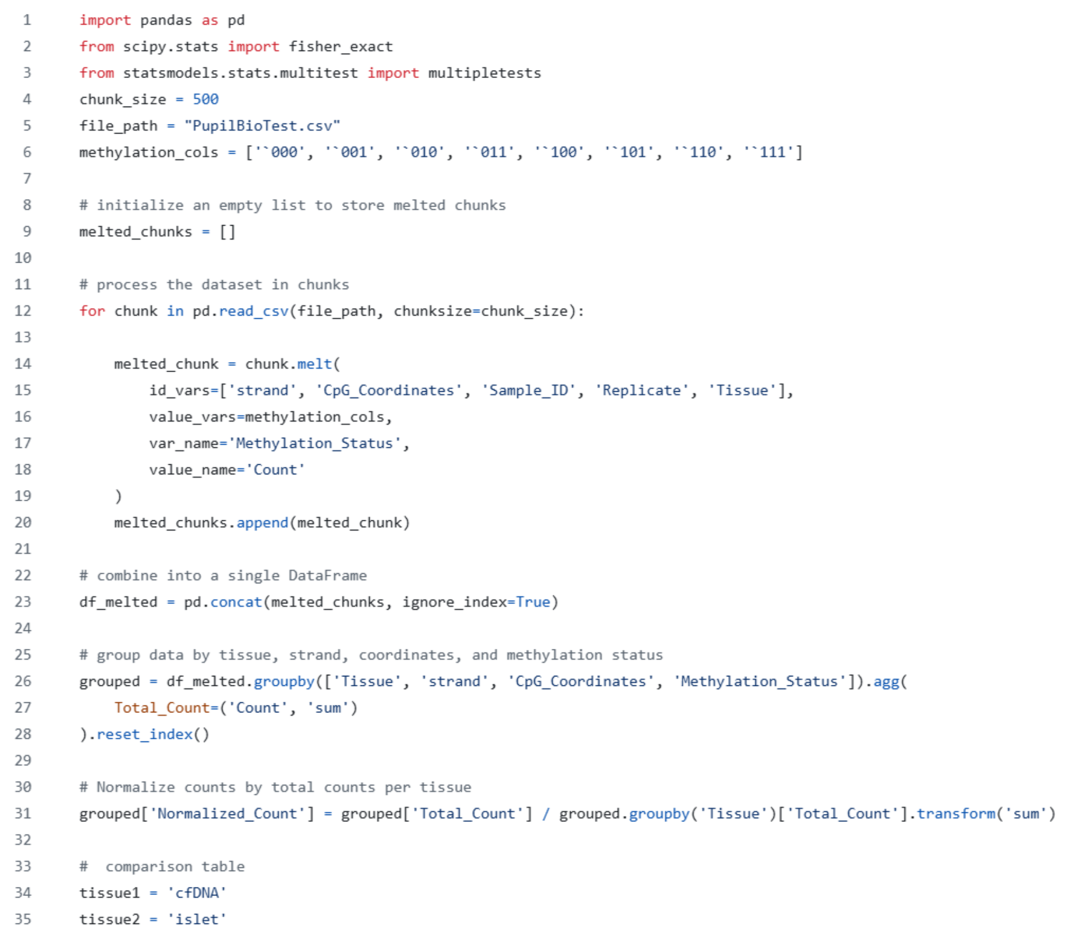


2. Biomarker Identification(**This analysis is done on the small part I am not able to do on whole provided csv file due to system limitations** **)**

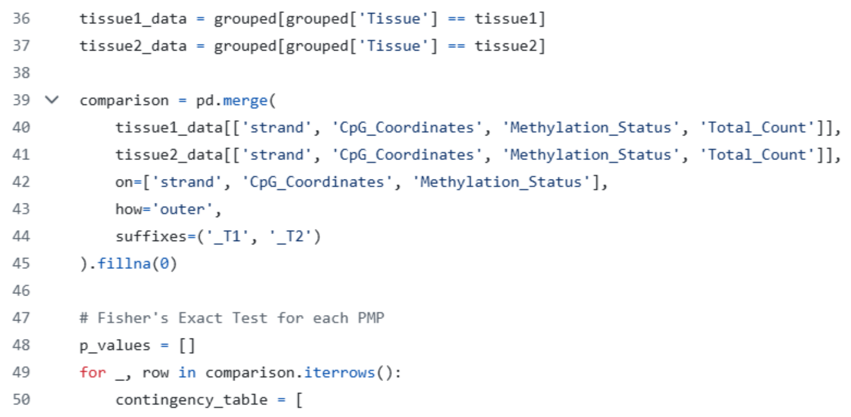
a. Identify PMPs with high specificity for tissue differentiation, minimizing false positives for Tissue #1 while allowing some false negatives. Use statistical or machine learning approaches to assign confidence (e.g., p-values) to each PMP.

Statistical method-

Codes-



```python
import pandas as pd
from scipy.stats import fisher_exact
from statsmodels.stats.multitest import multipletests
chunk_size = 500
file_path = "PupilBioTest.csv"
methylation_cols = ['`000', '`001', '`010', '`011', '`100', '`101', '`110', '`111']

# initialize an empty list to store melted chunks
melted_chunks = []

# process the dataset in chunks
for chunk in pd.read_csv(file_path, chunksize=chunk_size):

    melted_chunk = chunk.melt(
        id_vars=['strand', 'CpG_Coordinates', 'Sample_ID', 'Replicate', 'Tissue'],
        value_vars=methylation_cols,
        var_name='Methylation_Status',
        value_name='Count'
    )
    melted_chunks.append(melted_chunk)

# combine into a single DataFrame
df_melted = pd.concat(melted_chunks, ignore_index=True)

# group data by tissue, strand, coordinates, and methylation status
grouped = df_melted.groupby(['Tissue', 'strand', 'CpG_Coordinates', 'Methylation_Status']).agg(
    Total_Count=('Count', 'sum')
).reset_index()

# Normalize counts by total counts per tissue
grouped['Normalized_Count'] = grouped['Total_Count'] / grouped.groupby('Tissue')['Total_Count'].transform('sum')

#  comparison table
tissue1 = 'cfDNA'
tissue2 = 'islet'
```



```python
tissue1_data = grouped[grouped['Tissue'] == tissue1]
tissue2_data = grouped[grouped['Tissue'] == tissue2]

comparison = pd.merge(
    tissue1_data[['strand', 'CpG_Coordinates', 'Methylation_Status', 'Total_Count']],
    tissue2_data[['strand', 'CpG_Coordinates', 'Methylation_Status', 'Total_Count']],
    on=['strand', 'CpG_Coordinates', 'Methylation_Status'],
    how='outer',
    suffixes=('_T1', '_T2')
).fillna(0)

# Fisher's Exact Test for each PMP
p_values = []
for _, row in comparison.iterrows():
    contingency_table = [
            [row['Total_Count_T1'], sum(tissue1_data['Total_Count']) - row['Total_Count_T1']],
            [row['Total_Count_T2'], sum(tissue2_data['Total_Count']) - row['Total_Count_T2']],
        ]
        _, p = fisher_exact(contingency_table)
        p_values.append(p)

    comparison['P_Value'] = p_values

    # adusted p-values for multiple testing with error handling
    if not comparison['P_Value'].isnull().all() and len(comparison['P_Value']) > 0:
        # Remove NaN P-values and apply multiple testing correction
        valid_p_values = comparison['P_Value'].dropna()
        if len(valid_p_values) > 0:
            try:
                comparison['Adjusted_P_Value'] = multipletests(valid_p_values, method='fdr_bh')[1]
            except Exception as e:
                print(f"Error during multiple testing correction: {e}")
                comparison['Adjusted_P_Value'] = None
        else:
            print("Error: No valid P-values found for multiple testing correction.")
            comparison['Adjusted_P_Value'] = None
    else:
        print("Error: No valid P-values found for multiple testing correction.")
        comparison['Adjusted_P_Value'] = None

    # Filter PMPs
    significant_pmps = comparison[comparison['Adjusted_P_Value'] < 0.05]

    significant_pmps.to_csv('statistical_significant_pmps.csv', index=False)
    print("Significant PMPs based on statistical approach saved to 'statistical_significant_pmps.csv'")
```

<u>Result (sample) –</u>

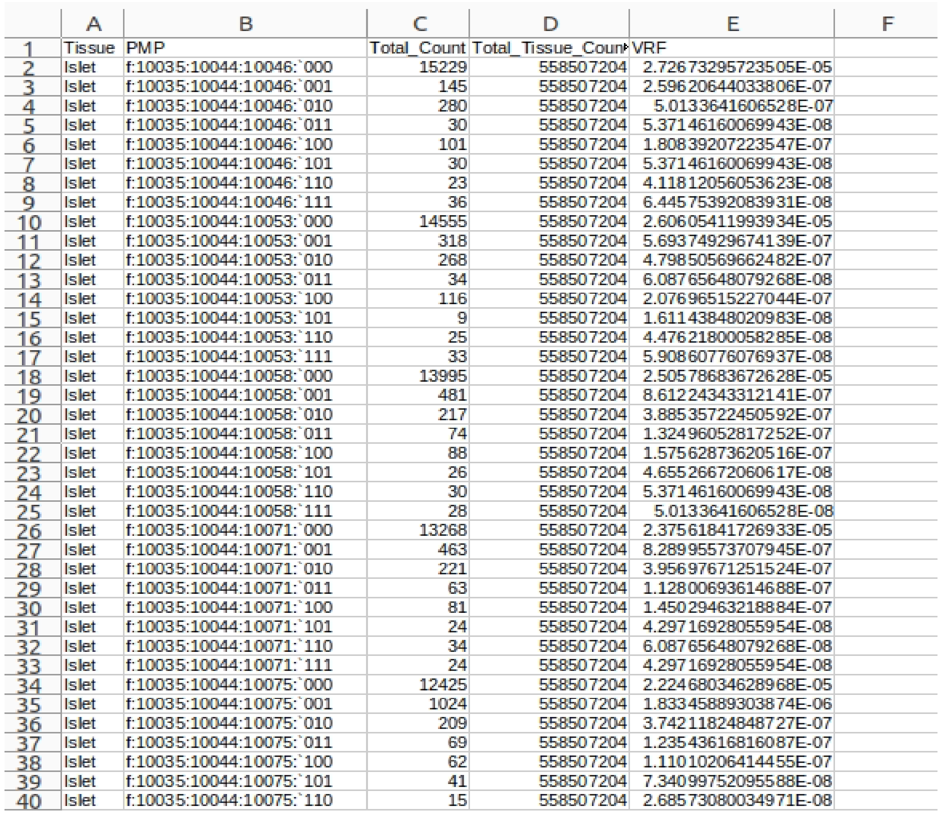
strand	CpG_Coordinates	Methylation_Status	Total_Count_T1	Total_Count_T2	P_Value	Adjusted_P_Value
r	10035:10044:10046	`000	1077.0	0.0	1.3382072844487038e-36	1.7676116401349276e-35
r	10035:10044:10053	`000	1086.0	0.0	5.917675284589969e-37	8.023259140671622e-36
r	10035:10044:10058	`000	1069.0	0.0	1.908798556018049e-36	2.497462623036202e-35
r	10035:10044:10071	`000	1072.0	0.0	1.9906808891810815e-36	2.5918875831200024e-35
r	10035:10044:10075	`000	1086.0	0.0	5.917675284589969e-37	8.023259140671622e-36
r	10035:10044:10081	`000	1086.0	0.0	5.917675284589969e-37	8.023259140671622e-36
r	10035:10044:10088	`000	1047.0	0.0	1.452516828653845e-35	1.810716014159768e-34
r	10035:10044:10096	`000	1056.0	0.0	6.371907791190341e-36	8.078305354230393e-35
r	10035:10044:10102	`000	1054.0	0.0	6.226306510017574e-36	7.951061266139723e-35
r	10035:10044:10109	`000	1005.0	0.0	3.281449035350466e-34	3.921801741714632e-33
r	10035:10044:10111	`000	1055.0	0.0	6.263405682991248e-36	7.969487437799825e-35
r	10035:10044:10118	`000	955.0	0.0	1.7386811904872064e-32	2.019134815267068e-31
r	10035:10044:10121	`000	907.0	0.0	5.7755991783838745e-31	6.59517144230489e-30
r	10035:10044:10130	`000	797.0	0.0	3.550894010064119e-27	3.7663965430886996e-26
r	10035:10044:10133	`000	764.0	0.0	3.604335951332221e-26	3.7630674200417176e-25
r	10035:10044:10140	`000	620.0	0.0	2.2639107299259707e-21	2.165189088302304e-20
r	10035:10044:10146	`000	450.0	0.0	1.6196031197187778e-15	1.396467889629982e-14

b. Calculate the mean variant read fraction (VRF) for each PMP in both tissues.

Code -

```python
import pandas as pd
file_path = "PupilBioTest.csv"  # Replace with your actual file name
methylation_cols = ['`000', '`001', '`010', '`011', '`100', '`101', '`110', '`111']
df = pd.read_csv(file_path)
df.columns = df.columns.str.strip()
required_columns = ['strand', 'CpG_Coordinates', 'Sample_ID', 'Replicate', 'Tissue'] + methylation_cols
for col in required_columns:
    if col not in df.columns:
        raise ValueError(f"Column '{col}' not found in the dataset. Please check your input file.")
# melt the dataset to create a 'Methylation_Status' column
df_melted = df.melt(
    id_vars=['strand', 'CpG_Coordinates', 'Sample_ID', 'Replicate', 'Tissue'],
    value_vars=methylation_cols,
    var_name='Methylation_Status',
    value_name='Count'
)
# Create PMP
df_melted['PMP'] = (
    df_melted['strand'] + ':' + df_melted['CpG_Coordinates'] + ':' + df_melted['Methylation_Status']
)
# group data by Tissue and PMP to calculate total counts
grouped = df_melted.groupby(['Tissue', 'PMP']).agg(
    Total_Count=('Count', 'sum')
).reset_index()
# Calculate the total counts for each tissue
grouped['Total_Tissue_Count'] = grouped.groupby('Tissue')['Total_Count'].transform('sum')
# Calculate VRF
grouped['VRF'] = grouped['Total_Count'] / grouped['Total_Tissue_Count']
output_file = "vrf_results.csv"
grouped.to_csv(output_file, index=False)
print(f"VRF results saved to {output_file}")
```

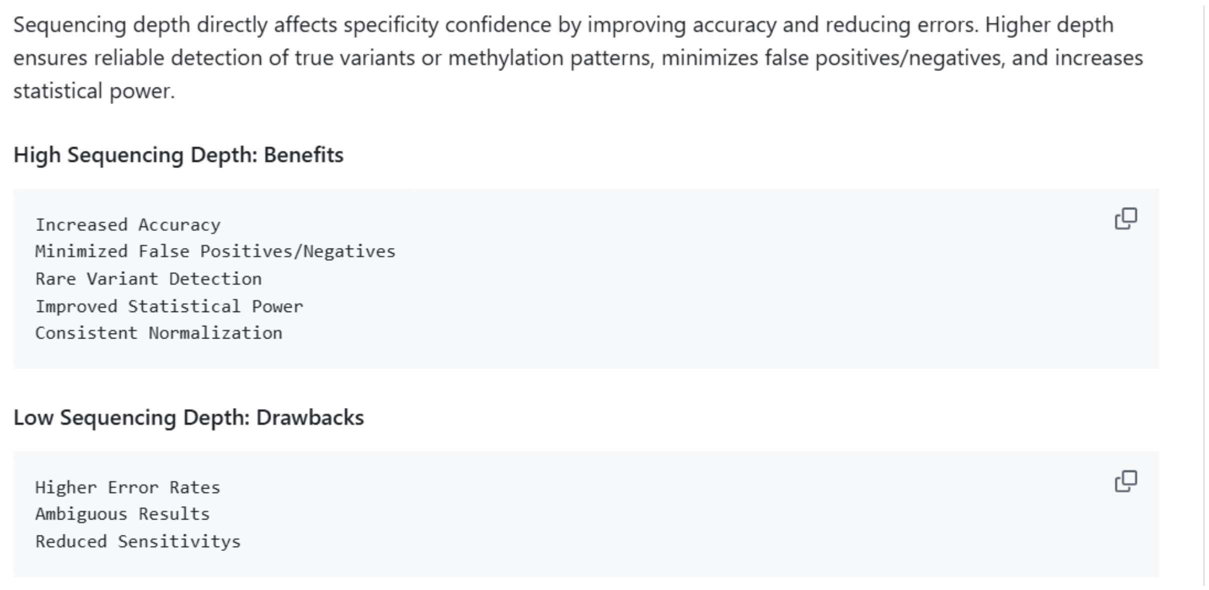
Result –



	A	B	C	D	E	F
1	Tissue	PMP	Total_Count	Total_Tissue_Coun	VRF	
2	Islet	f:10035:10044:10046:`000	15229	558507204	2.72673295723505E-05	
3	Islet	f:10035:10044:10046:`001	145	558507204	2.59620644033806E-07	
4	Islet	f:10035:10044:10046:`010	280	558507204	5.0133641606528E-07	
5	Islet	f:10035:10044:10046:`011	30	558507204	5.37146160069943E-08	
6	Islet	f:10035:10044:10046:`100	101	558507204	1.80839207223547E-07	
7	Islet	f:10035:10044:10046:`101	30	558507204	5.37146160069943E-08	
8	Islet	f:10035:10044:10046:`110	23	558507204	4.11812056053623E-08	
9	Islet	f:10035:10044:10046:`111	36	558507204	6.44575392083931E-08	
10	Islet	f:10035:10044:10053:`000	14555	558507204	2.60605411993934E-05	
11	Islet	f:10035:10044:10053:`001	318	558507204	5.69374929674139E-07	
12	Islet	f:10035:10044:10053:`010	268	558507204	4.79850569662482E-07	
13	Islet	f:10035:10044:10053:`011	34	558507204	6.08765648079268E-08	
14	Islet	f:10035:10044:10053:`100	116	558507204	2.07696515227044E-07	
15	Islet	f:10035:10044:10053:`101	9	558507204	1.61143848020983E-08	
16	Islet	f:10035:10044:10053:`110	25	558507204	4.47621800058285E-08	
17	Islet	f:10035:10044:10053:`111	33	558507204	5.90860776076937E-08	
18	Islet	f:10035:10044:10058:`000	13995	558507204	2.50578683672628E-05	
19	Islet	f:10035:10044:10058:`001	481	558507204	8.61224343312141E-07	
20	Islet	f:10035:10044:10058:`010	217	558507204	3.88535722450592E-07	
21	Islet	f:10035:10044:10058:`011	74	558507204	1.32496052817252E-07	
22	Islet	f:10035:10044:10058:`100	88	558507204	1.57562873620516E-07	
23	Islet	f:10035:10044:10058:`101	26	558507204	4.65526672060617E-08	
24	Islet	f:10035:10044:10058:`110	30	558507204	5.37146160069943E-08	
25	Islet	f:10035:10044:10058:`111	28	558507204	5.0133641606528E-08	
26	Islet	f:10035:10044:10071:`000	13268	558507204	2.37561841726933E-05	
27	Islet	f:10035:10044:10071:`001	463	558507204	8.28995573707945E-07	
28	Islet	f:10035:10044:10071:`010	221	558507204	3.95697671251524E-07	
29	Islet	f:10035:10044:10071:`011	63	558507204	1.12800693614688E-07	
30	Islet	f:10035:10044:10071:`100	81	558507204	1.45029463218884E-07	
31	Islet	f:10035:10044:10071:`101	24	558507204	4.29716928055954E-08	
32	Islet	f:10035:10044:10071:`110	34	558507204	6.08765648079268E-08	
33	Islet	f:10035:10044:10071:`111	24	558507204	4.29716928055954E-08	
34	Islet	f:10035:10044:10075:`000	12425	558507204	2.22468034628968E-05	
35	Islet	f:10035:10044:10075:`001	1024	558507204	1.83345889303874E-06	
36	Islet	f:10035:10044:10075:`010	209	558507204	3.74211824848727E-07	
37	Islet	f:10035:10044:10075:`011	69	558507204	1.23543616816087E-07	
38	Islet	f:10035:10044:10075:`100	62	558507204	1.11010206414455E-07	
39	Islet	f:10035:10044:10075:`101	41	558507204	7.34099752095588E-08	
40	Islet	f:10035:10044:10075:`110	15	558507204	2.68573080034971E-08	

3. Address the following questions

a. How does sequencing depth affect specificity confidence?



Sequencing depth directly affects specificity confidence by improving accuracy and reducing errors. Higher depth ensures reliable detection of true variants or methylation patterns, minimizes false positives/negatives, and increases statistical power.

High Sequencing Depth: Benefits

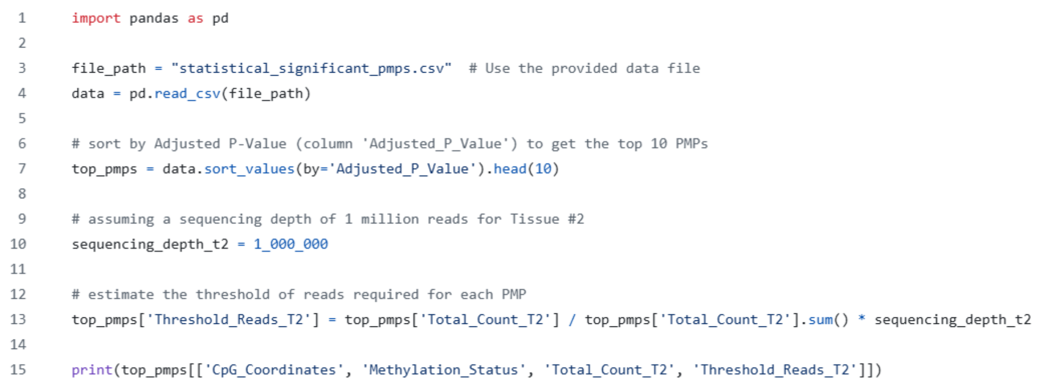
```
Increased Accuracy
Minimized False Positives/Negatives
Rare Variant Detection
Improved Statistical Power
Consistent Normalization
```

Low Sequencing Depth: Drawbacks

```
Higher Error Rates
Ambiguous Results
Reduced Sensitivitys
```

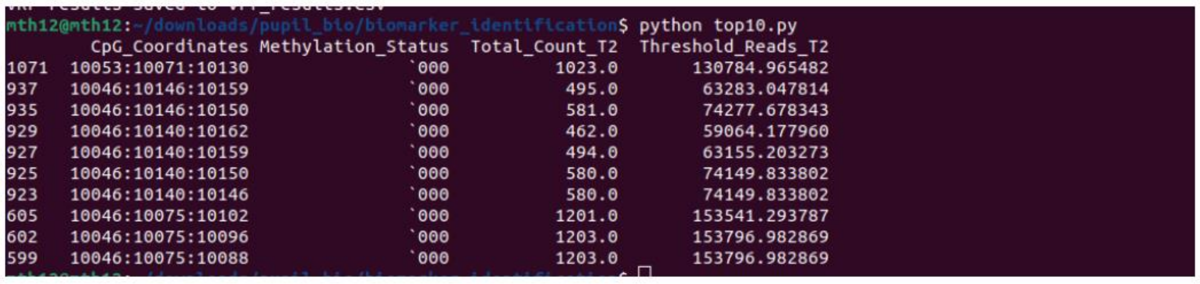
b. For the top 10 PMPs, estimate the threshold of reads required to confidently call Tissue #2 at a sequencing depth of 1 million reads.

Code-



```python
import pandas as pd

file_path = "statistical_significant_pmps.csv"  # Use the provided data file
data = pd.read_csv(file_path)

# sort by Adjusted P-Value (column 'Adjusted_P_Value') to get the top 10 PMPs
top_pmps = data.sort_values(by='Adjusted_P_Value').head(10)

# assuming a sequencing depth of 1 million reads for Tissue #2
sequencing_depth_t2 = 1_000_000

# estimate the threshold of reads required for each PMP
top_pmps['Threshold_Reads_T2'] = top_pmps['Total_Count_T2'] / top_pmps['Total_Count_T2'].sum() * sequencing_depth_t2

print(top_pmps[['CpG_Coordinates', 'Methylation_Status', 'Total_Count_T2', 'Threshold_Reads_T2']])
```

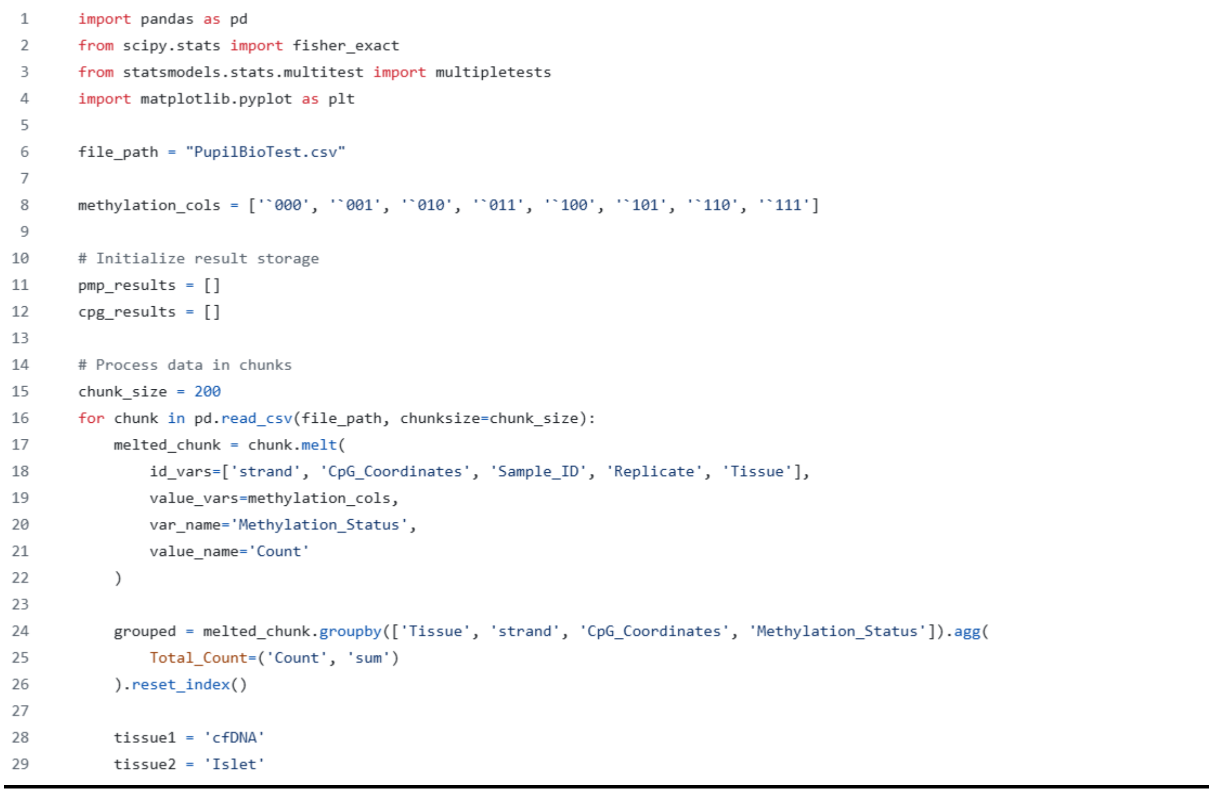
Result –



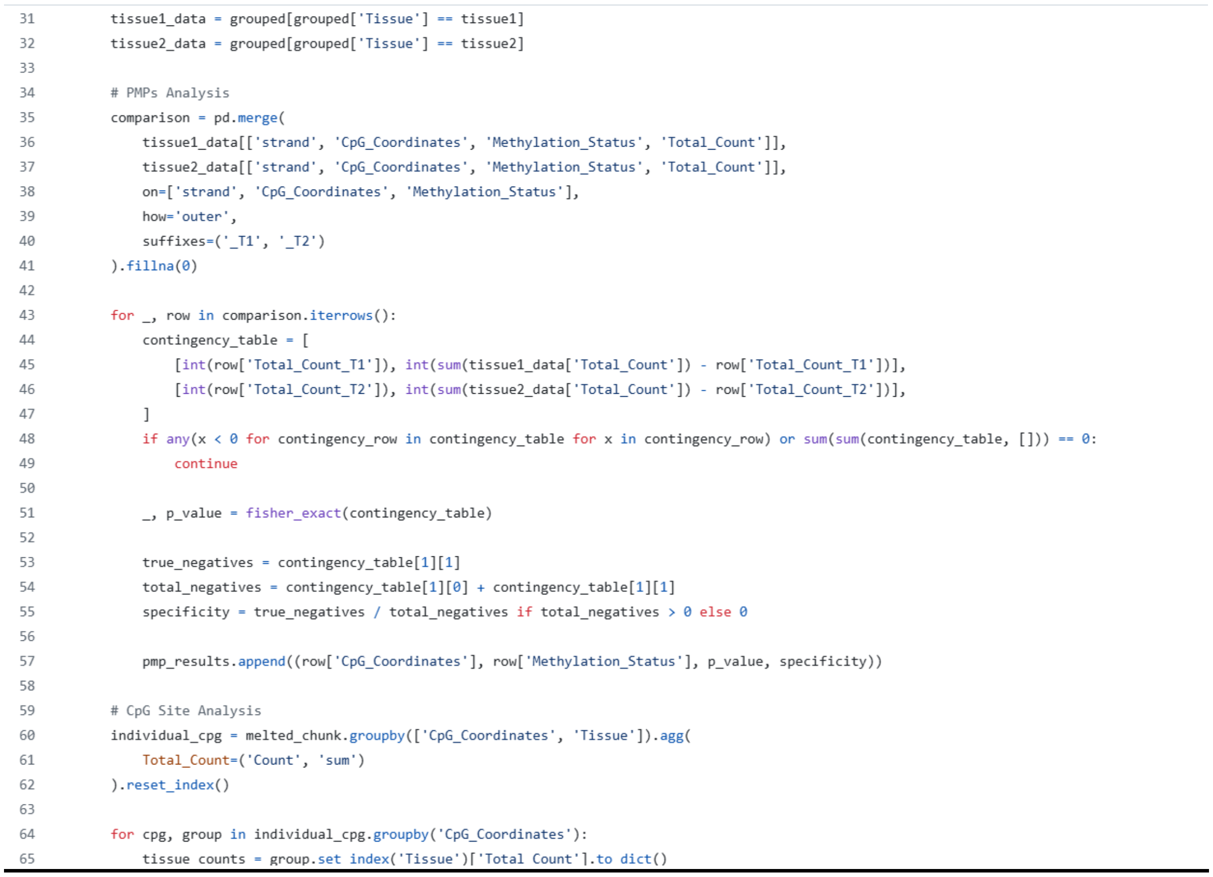
```
mth12@mth12:~/downloads/pupil_bio/biomarker_identification$ python top10.py
       CpG_Coordinates Methylation_Status  Total_Count_T2  Threshold_Reads_T2
1071  10053:10071:10130               `000          1023.0       130784.965482
937   10046:10146:10159               `000           495.0        63283.047814
935   10046:10146:10150               `000           581.0        74277.678343
929   10046:10140:10162               `000           462.0        59064.177960
927   10046:10140:10159               `000           494.0        63155.203273
925   10046:10140:10150               `000           580.0        74149.833802
923   10046:10140:10146               `000           580.0        74149.833802
605   10046:10075:10102               `000          1201.0       153541.293787
602   10046:10075:10096               `000          1203.0       153796.982869
599   10046:10075:10088               `000          1203.0       153796.982869
```

c. Validate the hypothesis by comparing the specificity of the top 10 PMPs against individual CpG sites.

Code-



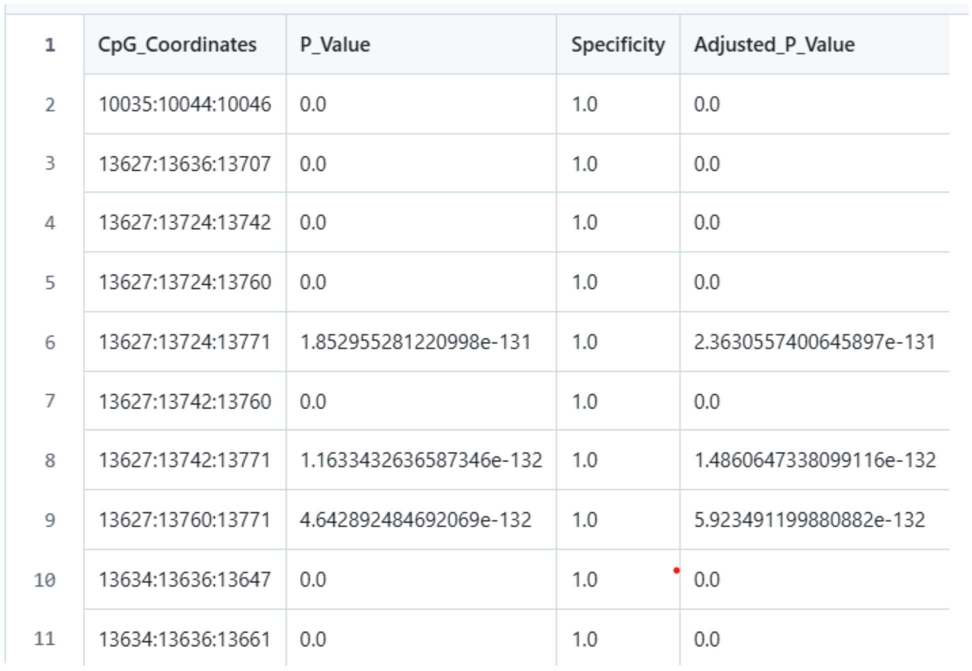
```python
import pandas as pd
from scipy.stats import fisher_exact
from statsmodels.stats.multitest import multipletests
import matplotlib.pyplot as plt

file_path = "PupilBioTest.csv"

methylation_cols = ['`000', '`001', '`010', '`011', '`100', '`101', '`110', '`111']

# Initialize result storage
pmp_results = []
cpg_results = []

# Process data in chunks
chunk_size = 200
for chunk in pd.read_csv(file_path, chunksize=chunk_size):
    melted_chunk = chunk.melt(
        id_vars=['strand', 'CpG_Coordinates', 'Sample_ID', 'Replicate', 'Tissue'],
        value_vars=methylation_cols,
        var_name='Methylation_Status',
        value_name='Count'
    )

    grouped = melted_chunk.groupby(['Tissue', 'strand', 'CpG_Coordinates', 'Methylation_Status']).agg(
        Total_Count=('Count', 'sum')
    ).reset_index()

    tissue1 = 'cfDNA'
    tissue2 = 'Islet'
```



```python
    tissue1_data = grouped[grouped['Tissue'] == tissue1]
    tissue2_data = grouped[grouped['Tissue'] == tissue2]

    # PMPs Analysis
    comparison = pd.merge(
        tissue1_data[['strand', 'CpG_Coordinates', 'Methylation_Status', 'Total_Count']],
        tissue2_data[['strand', 'CpG_Coordinates', 'Methylation_Status', 'Total_Count']],
        on=['strand', 'CpG_Coordinates', 'Methylation_Status'],
        how='outer',
        suffixes=('_T1', '_T2')
    ).fillna(0)

    for _, row in comparison.iterrows():
        contingency_table = [
            [int(row['Total_Count_T1']), int(sum(tissue1_data['Total_Count']) - row['Total_Count_T1'])],
            [int(row['Total_Count_T2']), int(sum(tissue2_data['Total_Count']) - row['Total_Count_T2'])],
        ]
        if any(x < 0 for contingency_row in contingency_table for x in contingency_row) or sum(sum(contingency_table, [])) == 0:
            continue

        _, p_value = fisher_exact(contingency_table)

        true_negatives = contingency_table[1][1]
        total_negatives = contingency_table[1][0] + contingency_table[1][1]
        specificity = true_negatives / total_negatives if total_negatives > 0 else 0

        pmp_results.append((row['CpG_Coordinates'], row['Methylation_Status'], p_value, specificity))

    # CpG Site Analysis
    individual_cpg = melted_chunk.groupby(['CpG_Coordinates', 'Tissue']).agg(
        Total_Count=('Count', 'sum')
    ).reset_index()

    for cpg, group in individual_cpg.groupby('CpG_Coordinates'):
        tissue_counts = group.set_index('Tissue')['Total_Count'].to_dict()
        count_t1 = tissue_counts.get(tissue1, 0)
        count_t2 = tissue_counts.get(tissue2, 0)
        contingency_table = [
            [count_t1, sum(group['Total_Count']) - count_t1],
            [count_t2, sum(group['Total_Count']) - count_t2],
        ]
        if any(x < 0 for contingency_row in contingency_table for x in contingency_row) or sum(sum(contingency_table, [])) == 0:
            continue

        _, p_value = fisher_exact(contingency_table)

        true_negatives = contingency_table[1][1]
        total_negatives = contingency_table[1][0] + contingency_table[1][1]
        specificity = true_negatives / total_negatives if total_negatives > 0 else 0
        cpg_results.append((cpg, p_value, specificity))
# Convert results to DataFrame
pmp_df = pd.DataFrame(pmp_results, columns=['CpG_Coordinates', 'Methylation_Status', 'P_Value', 'Specificity'])
cpg_df = pd.DataFrame(cpg_results, columns=['CpG_Coordinates', 'P_Value', 'Specificity'])
# Adjust p-values
pmp_df['Adjusted_P_Value'] = multipletests(pmp_df['P_Value'], method='fdr_bh')[1]
cpg_df['Adjusted_P_Value'] = multipletests(cpg_df['P_Value'], method='fdr_bh')[1]
# Filter top 10 PMPs and CpG sites
top_pmps = pmp_df.sort_values(by='Specificity', ascending=False).head(10)
top_cpgs = cpg_df.sort_values(by='Specificity', ascending=False).head(10)
# Compare Specificity Distributions
plt.boxplot([top_pmps['Specificity'], top_cpgs['Specificity']], labels=['PMPs', 'CpG Sites'])
plt.ylabel('Specificity')
plt.title('Specificity Comparison: PMPs vs CpG Sites')
plt.show()
top_pmps.to_csv("top_10_pmps.csv", index=False)
top_cpgs.to_csv("top_10_cpg_sites.csv", index=False)
print("Top 10 PMPs and CpG sites saved. Specificity comparison completed.")
```

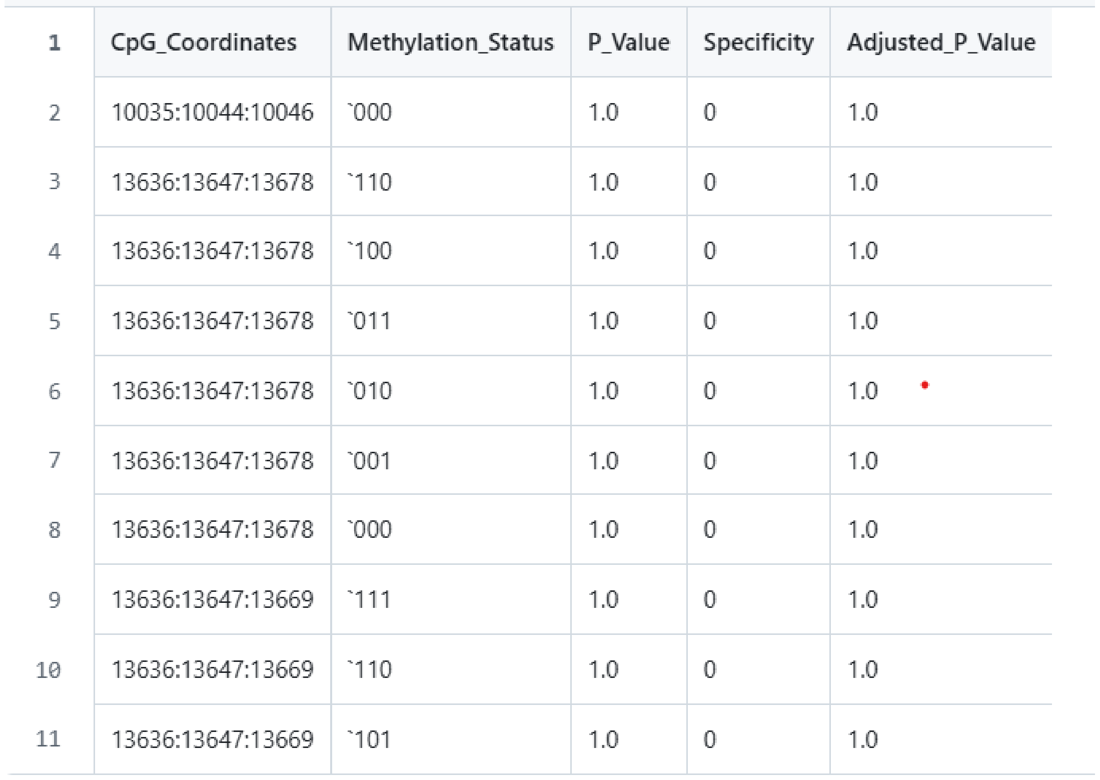
<u>Result –</u>

Top 10 cpg sites



CpG_Coordinates	P_Value	Specificity	Adjusted_P_Value
10035:10044:10046	0.0	1.0	0.0
13627:13636:13707	0.0	1.0	0.0
13627:13724:13742	0.0	1.0	0.0
13627:13724:13760	0.0	1.0	0.0
13627:13724:13771	1.852955281220998e-131	1.0	2.3630557400645897e-131
13627:13742:13760	0.0	1.0	0.0
13627:13742:13771	1.1633432636587346e-132	1.0	1.4860647338099116e-132
13627:13760:13771	4.642892484692069e-132	1.0	5.923491199880882e-132
13634:13636:13647	0.0	1.0	0.0
13634:13636:13661	0.0	1.0	0.0

Top 10 PMP’S



CpG_Coordinates	Methylation_Status	P_Value	Specificity	Adjusted_P_Value
10035:10044:10046	`000	1.0	0	1.0
13636:13647:13678	`110	1.0	0	1.0
13636:13647:13678	`100	1.0	0	1.0
13636:13647:13678	`011	1.0	0	1.0
13636:13647:13678	`010	1.0	0	1.0
13636:13647:13678	`001	1.0	0	1.0
13636:13647:13678	`000	1.0	0	1.0
13636:13647:13669	`111	1.0	0	1.0
13636:13647:13669	`110	1.0	0	1.0
13636:13647:13669	`101	1.0	0	1.0

The analysis shows that individual CpG sites have perfect specificity (1.0), while PMPs have very low specificity (0.0). Although CpG sites demonstrate high reliability in this study, their use as biomarkers is limited by errors from bisulfite sequencing, sampling techniques, and biological variability. PMPs aim to overcome these issues but currently fail to achieve sufficient specificity. Future work should focus on improving PMP design or integrating CpG site reliability with PMP robustness for better epigenetic biomarkers.

Task 2: NGS Data Analysis

Objective: Evaluate candidates' ability to process and analyze raw sequencing data.

Executive Summary

The NGS data analysis encompasses a series of steps to evaluate and interpret sequencing data . Initially, raw sequencing data undergo quality control (QC) with FastQC and MultiQC to assess data quality and identify issues such as adapter contamination. Adapter trimming and additional QC are performed as necessary to refine read quality. Next, the GRCh38 reference genome is indexed using BWA, followed by alignment of reads to the genome to map sequences accurately.

Sub-tasks:

1. Quality Control:

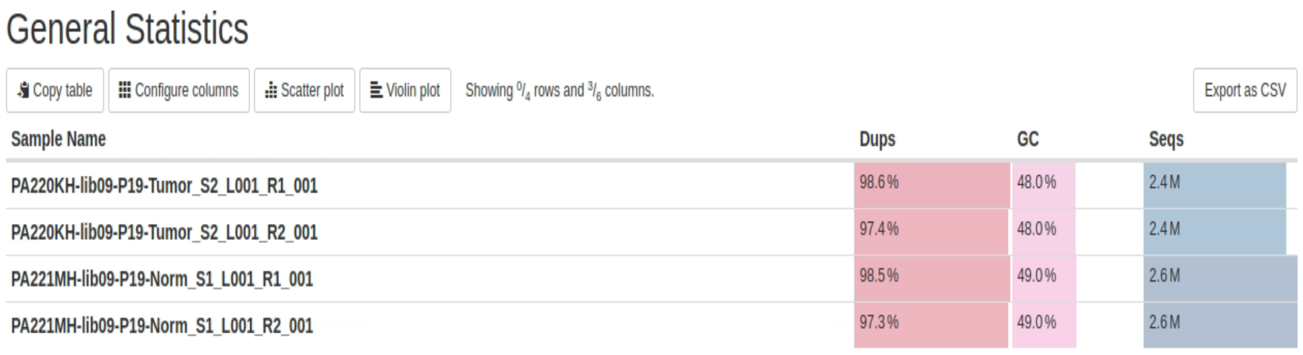
a.

i. Perform quality control using FastQC to evaluate the quality of the data.
ii. Provide a summary report using MultiQC with key quality metrics such as sequence count per sample, per base sequence quality, read length distribution, sequence duplication level and any adapter contamination.

Quality control - The quality control process for sequencing data involves using FastQC to assess key metrics such as sequence quality, read length distribution, and adapter contamination. MultiQC consolidates these results, providing an overview of quality metrics across samples. This ensures the data's reliability for subsequent analysis.

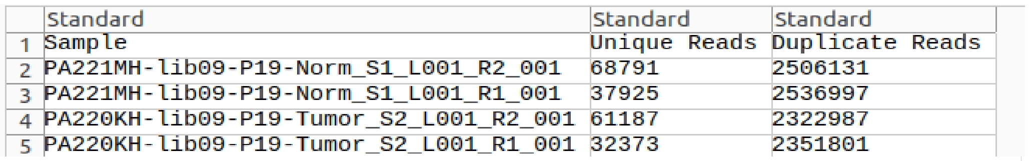
Tool: FastQC (https://github.com/s-andrews/FastQC)
Command: `fastqc <filename.fastq> -o <output_directory>`

Fig 1: General Statistics of Quality Control



Sample Name	Dups	GC	Seqs
PA220KH-lib09-P19-Tumor_S2_L001_R1_001	98.6%	48.0%	2.4M
PA220KH-lib09-P19-Tumor_S2_L001_R2_001	97.4%	48.0%	2.4M
PA221MH-lib09-P19-Norm_S1_L001_R1_001	98.5%	49.0%	2.6M
PA221MH-lib09-P19-Norm_S1_L001_R2_001	97.3%	49.0%	2.6M

Fig 2: Fastqc sequence counts plot



	Standard	Standard	Standard
1	Sample	Unique Reads	Duplicate Reads
2	PA221MH-lib09-P19-Norm_S1_L001_R2_001	68791	2506131
3	PA221MH-lib09-P19-Norm_S1_L001_R1_001	37925	2536997
4	PA220KH-lib09-P19-Tumor_S2_L001_R2_001	61187	2322987
5	PA220KH-lib09-P19-Tumor_S2_L001_R1_001	32373	2351801

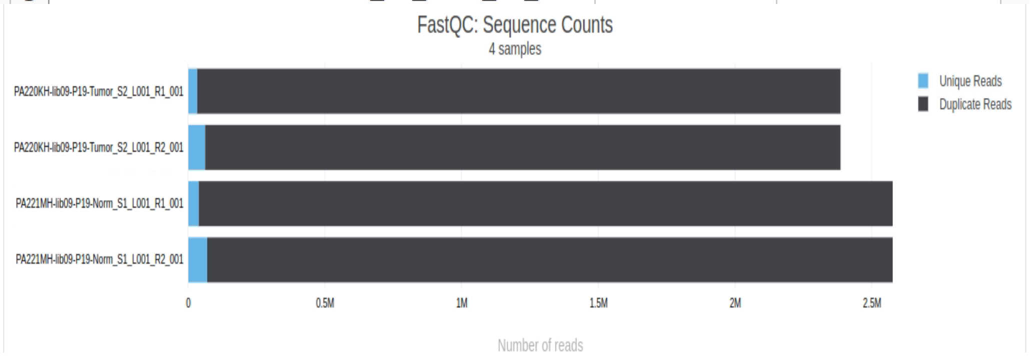


Fig 3: Mean Sequence Quality

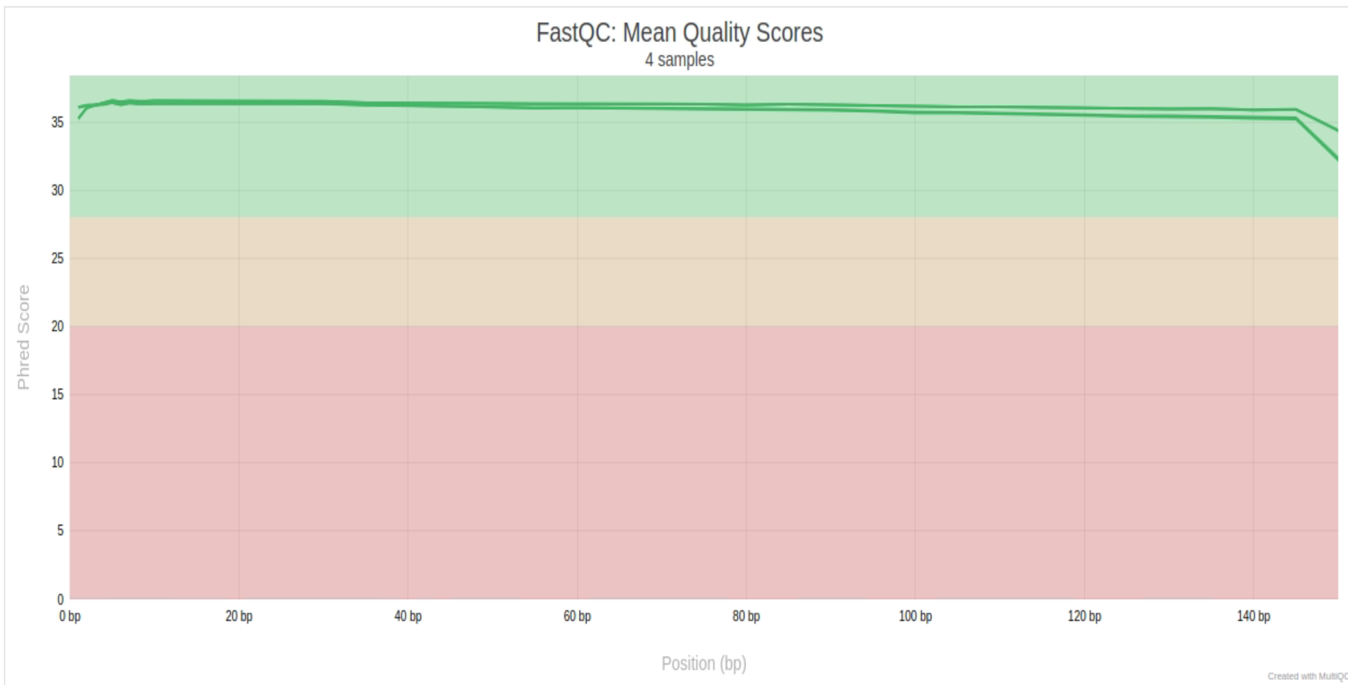
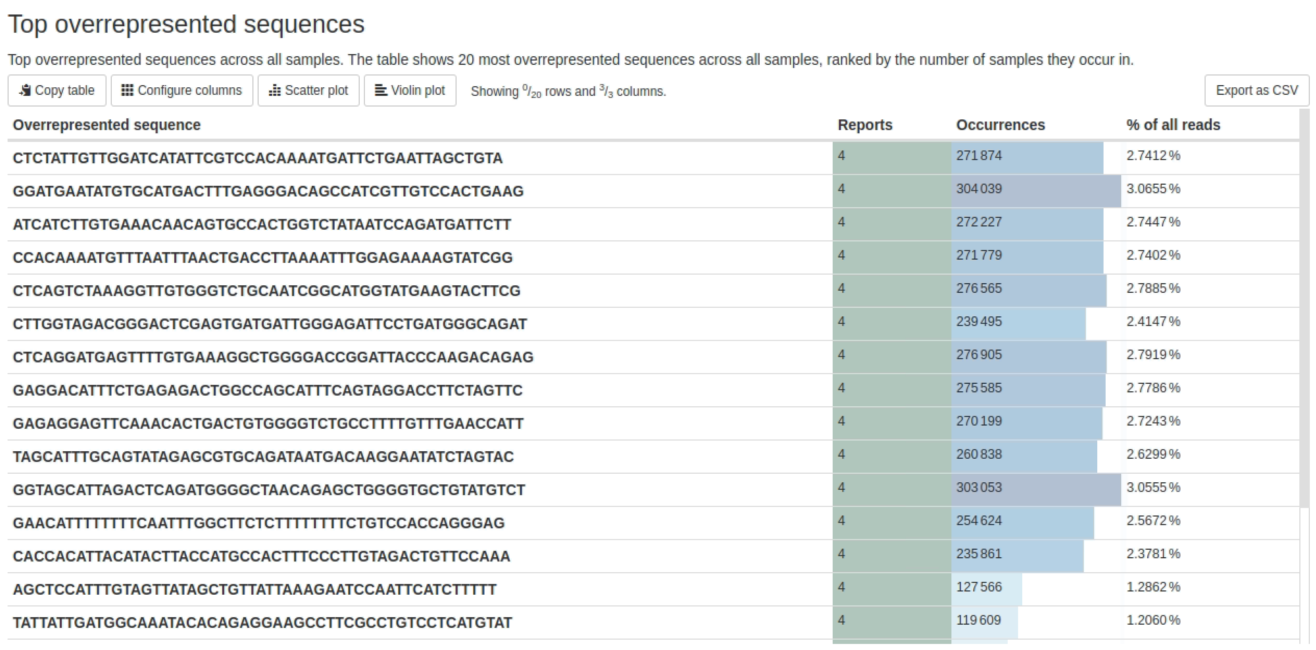


Fig 4: Top Overrepresented Sequences



Top overrepresented sequences

Top overrepresented sequences across all samples. The table shows 20 most overrepresented sequences across all samples, ranked by the number of samples they occur in.

Copy table
Configure columns
Scatter plot
Violin plot
Showing $^{0}/_{20}$ rows and $^{3}/_{3}$ columns.
Export as CSV

Overrepresented sequence	Reports	Occurrences	% of all reads
CTCTATTGTTGGATCATATTCGTCCACAAAATGATTCTGAATTAGCTGTA	4	271 874	2.7412 %
GGATGAATATGTGCATGACTTTGAGGGACAGCCATCGTTGTCCACTGAAG	4	304 039	3.0655 %
ATCATCTTGTGAAACAACAGTGCCACTGGTCTATAATCCAGATGATTCTT	4	272 227	2.7447 %
CCACAAAATGTTTAATTTAACTGACCTTAAAATTTGGAGAAAAGTATCGG	4	271 779	2.7402 %
CTCAGTCTAAAGGTTGTGGGTCTGCAATCGGCATGGTATGAAGTACTTCG	4	276 565	2.7885 %
CTTGGTAGACGGGACTCGAGTGATGATTGGGAGATTCCTGATGGGCAGAT	4	239 495	2.4147 %
CTCAGGATGAGTTTTGTGAAAGGCTGGGGACCGGATTACCCAAGACAGAG	4	276 905	2.7919 %
GAGGACATTTCTGAGAGACTGGCCAGCATTTCAGTAGGACCTTCTAGTTC	4	275 585	2.7786 %
GAGAGGAGTTCAAACACTGACTGTGGGGTCTGCCTTTTGTTTGAACCATT	4	270 199	2.7243 %
TAGCATTTGCAGTATAGAGCGTGCAGATAATGACAAGGAATATCTAGTAC	4	260 838	2.6299 %
GGTAGCATTAGACTCAGATGGGGCTAACAGAGCTGGGGTGCTGTATGTCT	4	303 053	3.0555 %
GAACATTTTTTTTCAATTTGGCTTCTCTTTTTTTTCTGTCCACCAGGGAG	4	254 624	2.5672 %
CACCACATTACATACTTACCATGCCACTTTCCCTTGTAGACTGTTCCAAA	4	235 861	2.3781 %
AGCTCCATTTGTAGTTATAGCTGTTATTAAAGAATCCAATTCATCTTTTT	4	127 566	1.2862 %
TATTATTGATGGCAAATACACAGAGGAAGCCTTCGCCTGTCCTCATGTAT	4	119 609	1.2060 %

Fig 5: Per Sequence Quality Scores

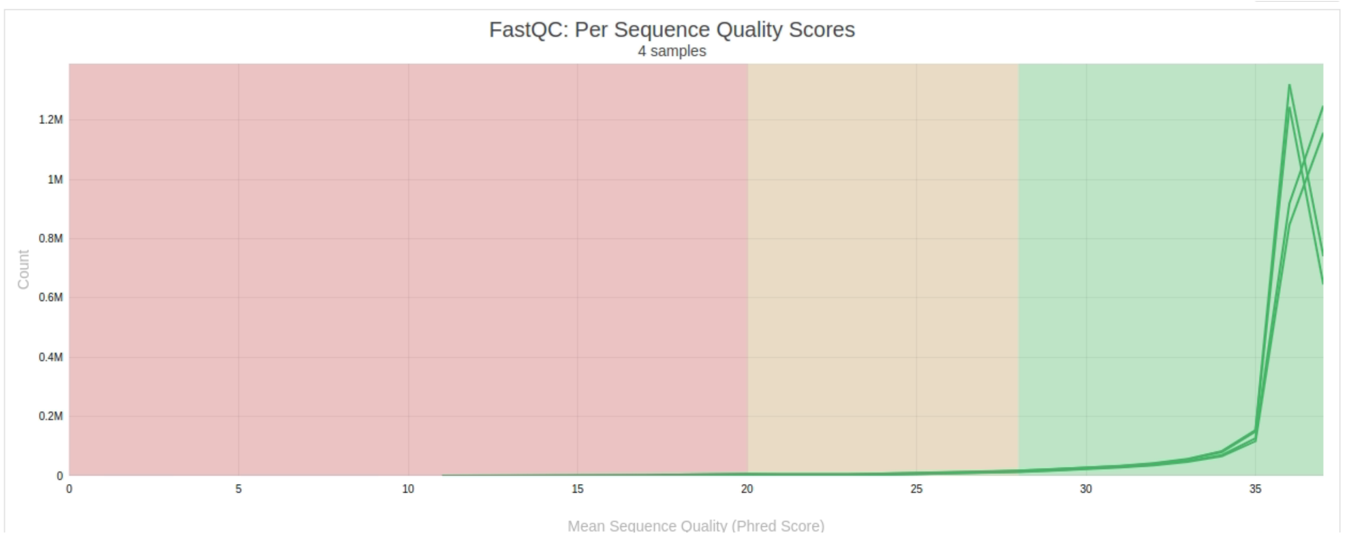


Fig 6: Sequence Duplication Levels

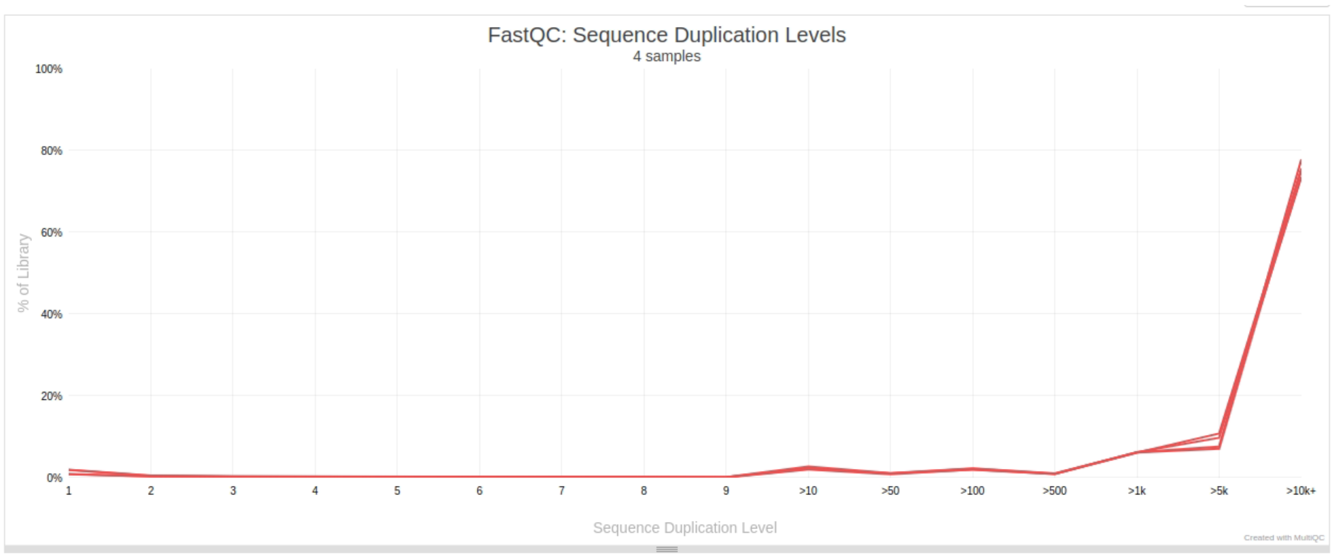
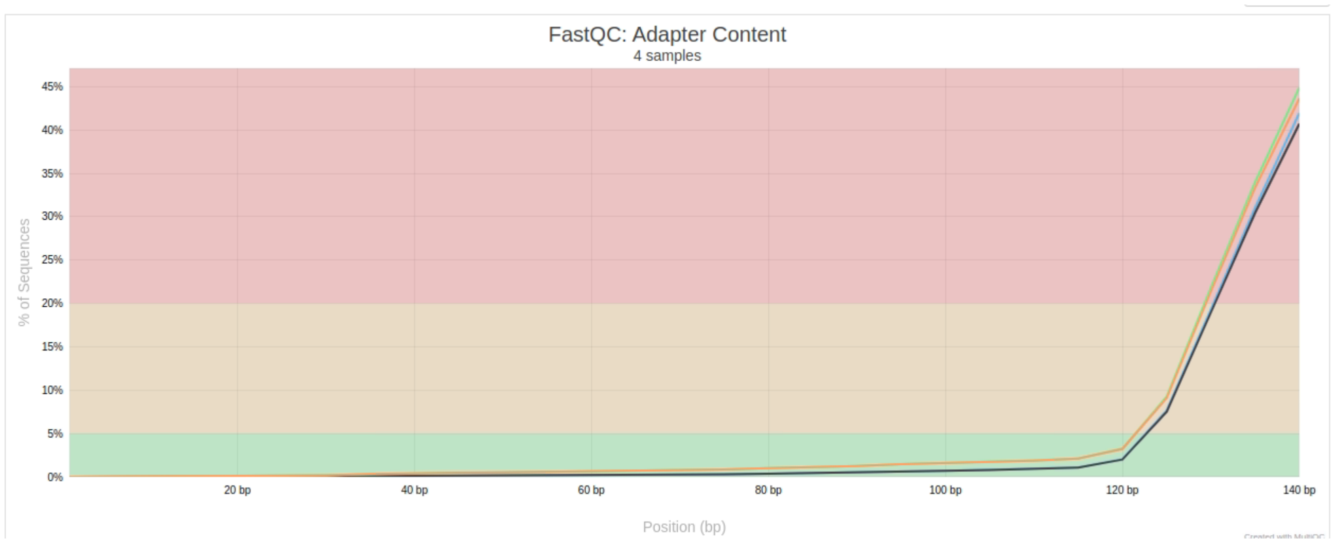


Fig 7:Adapter Content



Adapter Trimming:

i. If the data shows signs of adapter contamination or poor quality, perform trimming using Cutadapt, fastp or other suitable preprocessing and quality control software.
ii. Provide a summary report detailing the percentage of reads trimmed and any improvement in quality metrics after trimming.

Adapter Trimming: We will perform adapter trimming to eliminate the illumina universal adapters sequences and low-quality bases Using **Trim Galore default Parameters**, we aim to enhance the overall quality of the reads.

Tool - Trim galore (https://github.com/FelixKrueger/TrimGalore)
Command – `trim_galore --paired <read1.fastq> <read2.fastq>`

Fig 8:Lengths of trimmed sequences(3'end)

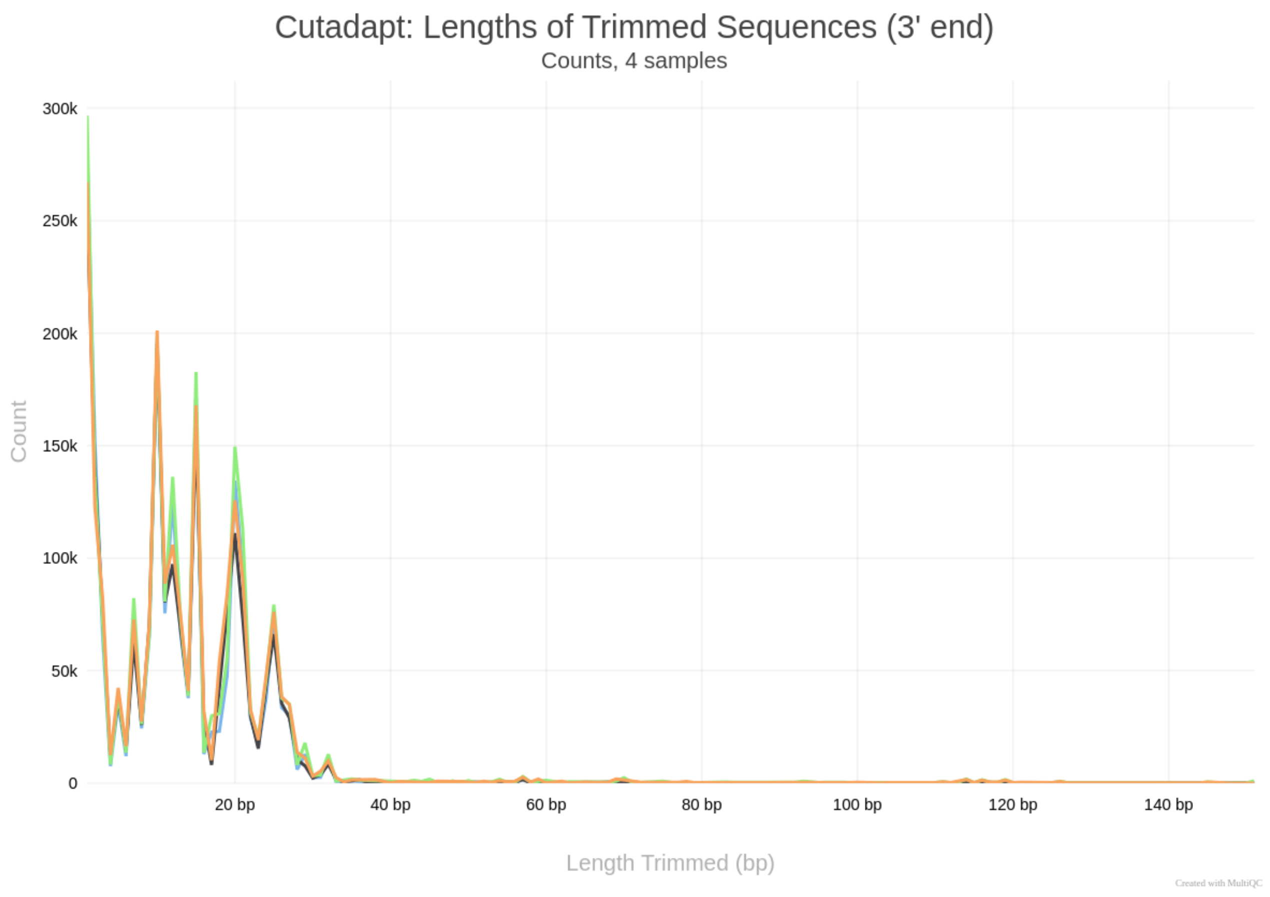
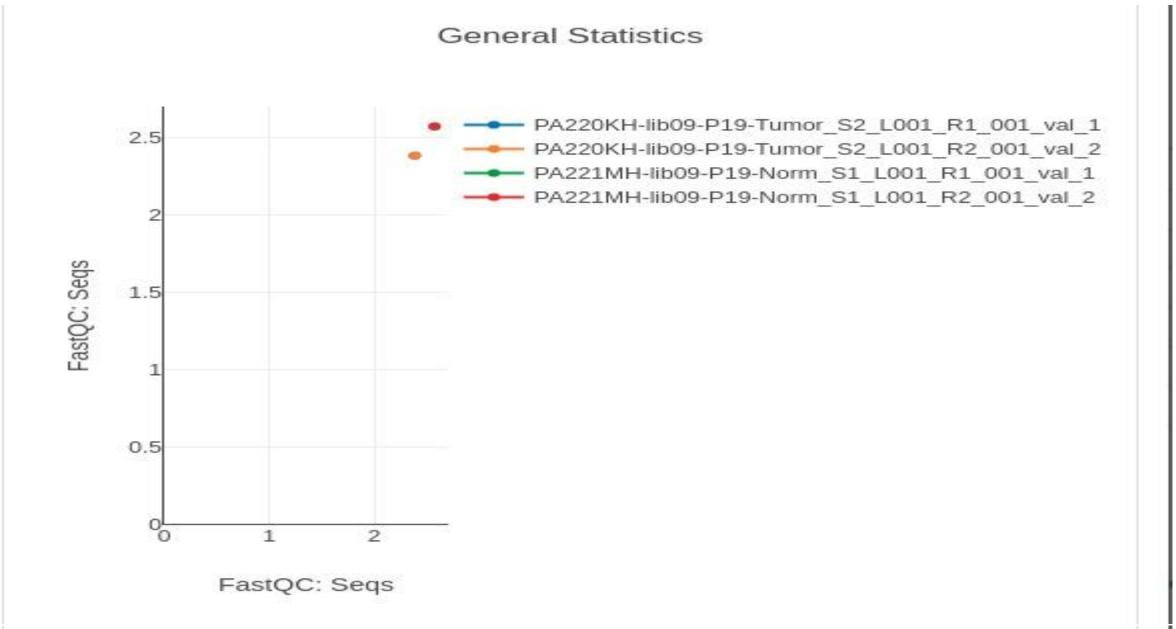
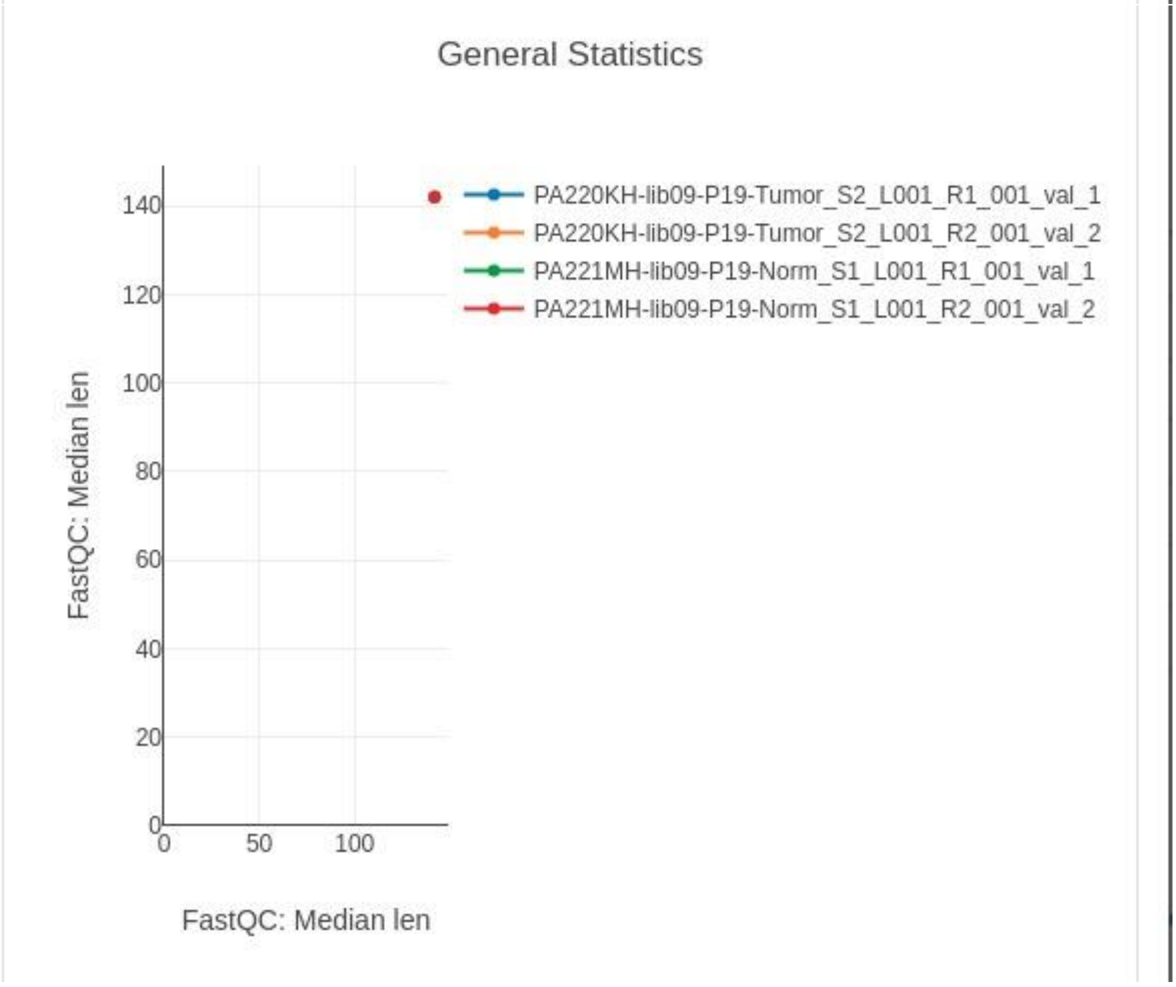


Fig 9: General statistics





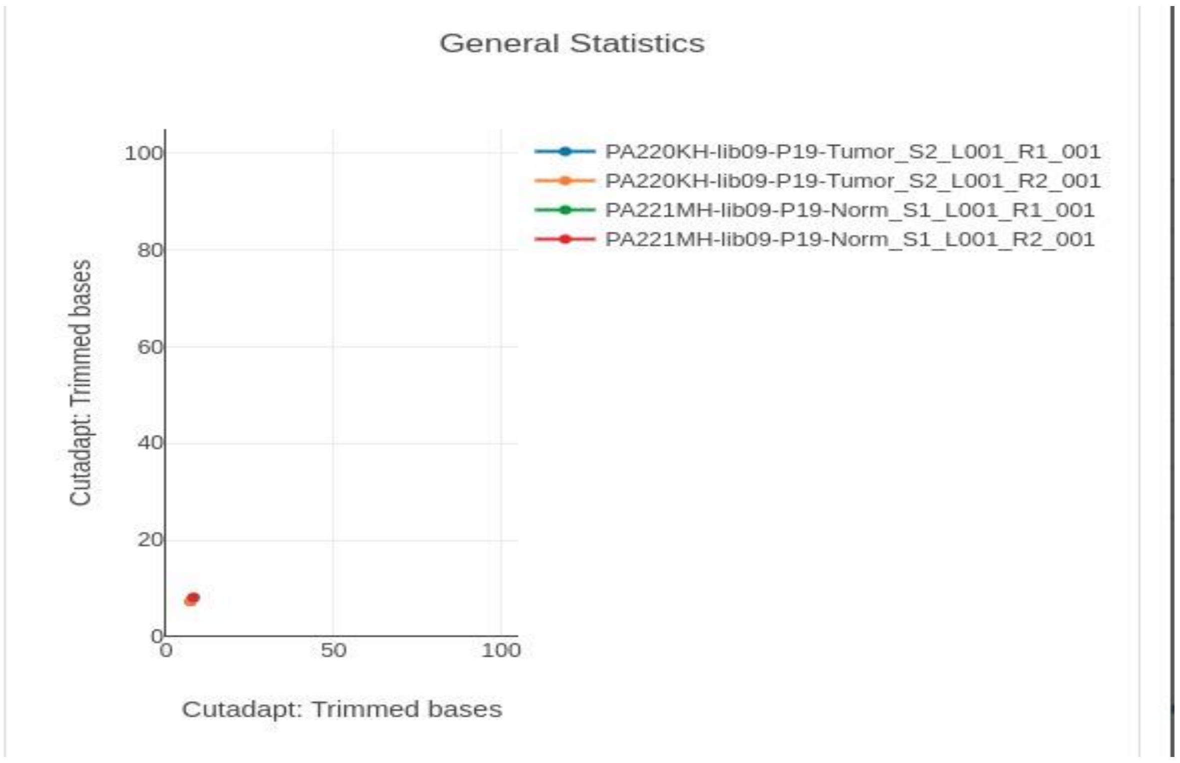


Table1: General Statistics

Sample	**Trimmed bases**	**Dups**	**GC**	**Seqs**
PA220KH-lib09-P19-Tumor_S2_L001_R1_001	7.2982011618924	.	.	.
PA220KH-lib09-P19-Tumor_S2_L001_R1_001_val_1	.	98.6081746452989	48	2.383345
PA220KH-lib09-P19-Tumor_S2_L001_R2_001	7.35042994911862	.	.	.
PA220KH-lib09-P19-Tumor_S2_L001_R2_001_val_2	.	97.405746964875	48	2.383345
PA221MH-lib09-P19-Norm_S1_L001_R1_001	8.15580031895109	.	.	.
PA221MH-lib09-P19-Norm_S1_L001_R1_001_val_1	.	98.4689852171313	48	2.571693
PA221MH-lib09-P19-Norm_S1_L001_R2_001	8.15324845100047	.	.	.
PA221MH-lib09-P19-Norm_S1_L001_R2_001_val_2	.	97.3352184728115	48	2.571693

2. Alignment and Mutation Calling

a. Align the samples to the human genome using tools like Bowtie2 or BWA

Genome Preparation

i. Prepare a genome index using BWA or other suitable alignment software for the provided genome reference**.**

Genome Preparation: GRCh38 Reference genome index was prepared using BWA. Indexing the reference genome allows the alignment tool to quickly locate matching sequences during analysis, enhancing processing speed and accuracy. BWA generates a series of indexed files based on the reference genome, which are then used in downstream alignment steps to map sequencing reads with high precision.

Tool - BWA (https://github.com/lh3/bwa)

Command – `bwa index reference_genome.fasta`

Alignment and Mapping:

ii. Perform read alignment using BWA.

iii. Provide alignment statistics and its visualization report, including the percentage of aligned reads, mapped reads, and potential issues with multi-mapping.

Mapping: Using BWA, we aligned sequencing reads to the GRCh38 reference genome. This alignment allows for validation of sequencing reads.

Tool - BWA

Command -
```
bwa mem reference_genome.fasta sample_1.fastq sample_2.fastq > aligned_reads.sam
```

Table 2: General Statistics

Sample	**Reads**	**Reads mapped**	**% Reads mapped**
PA220KH-lib09-P19-Tumor_S2_L001	4.829947	4.820164	99.8
PA221MH-lib09-P19-Norm_S1_L001	5.228701	5.217264	99.78

Fig 10: Samtools Flagstat

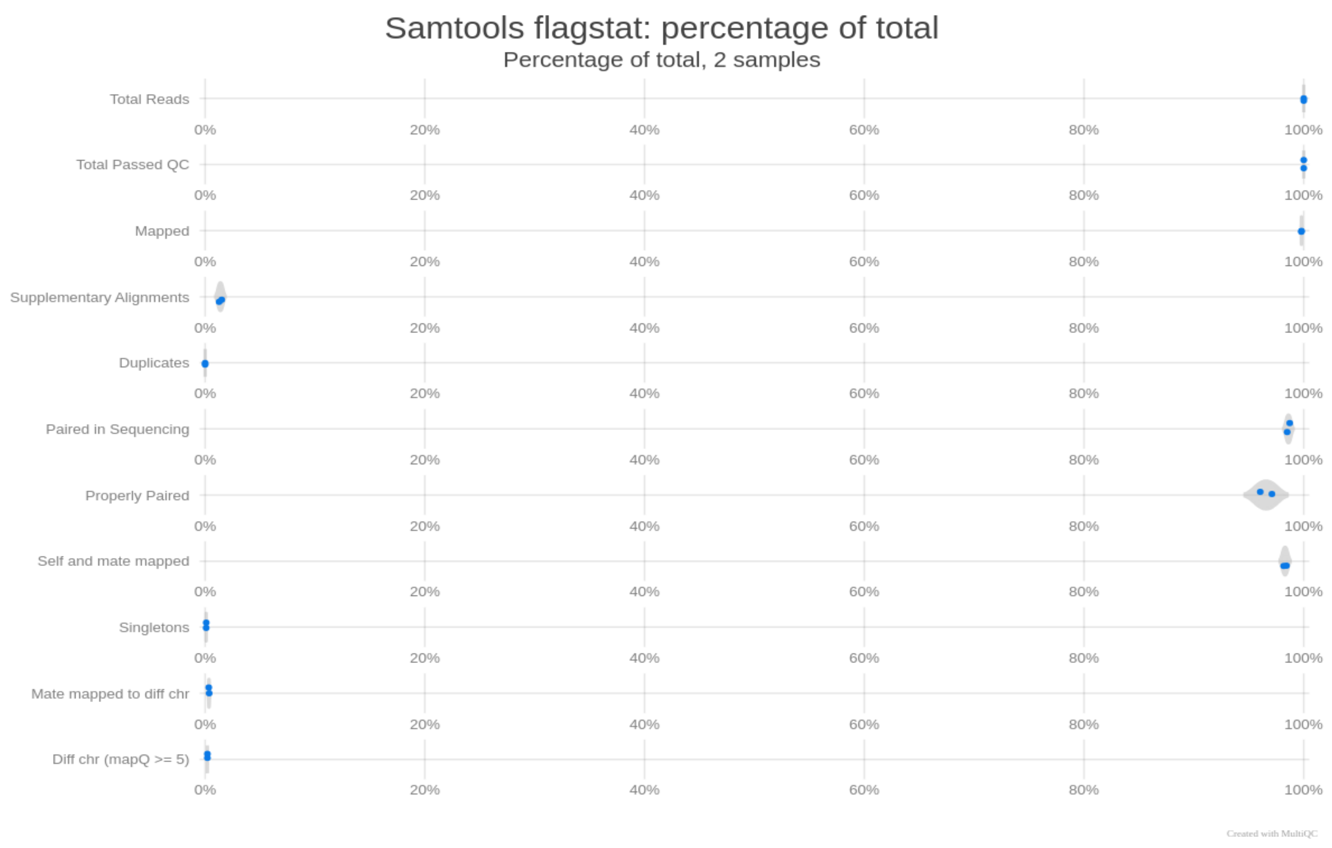
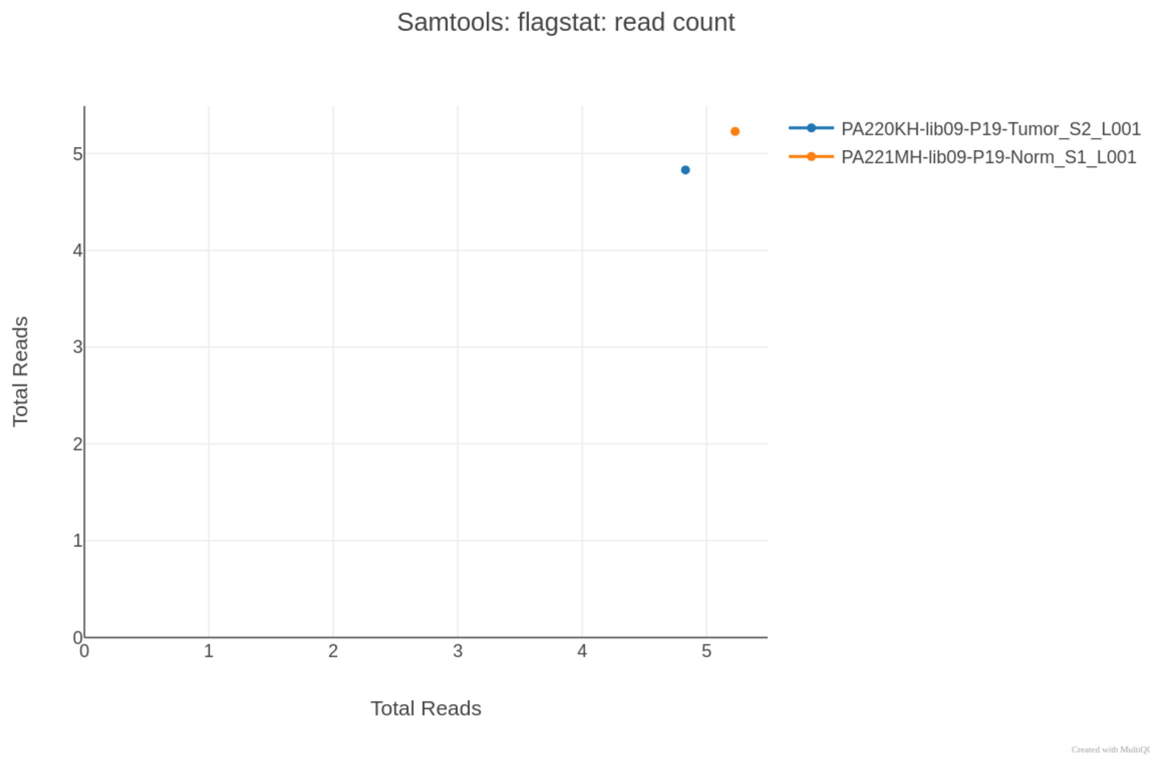
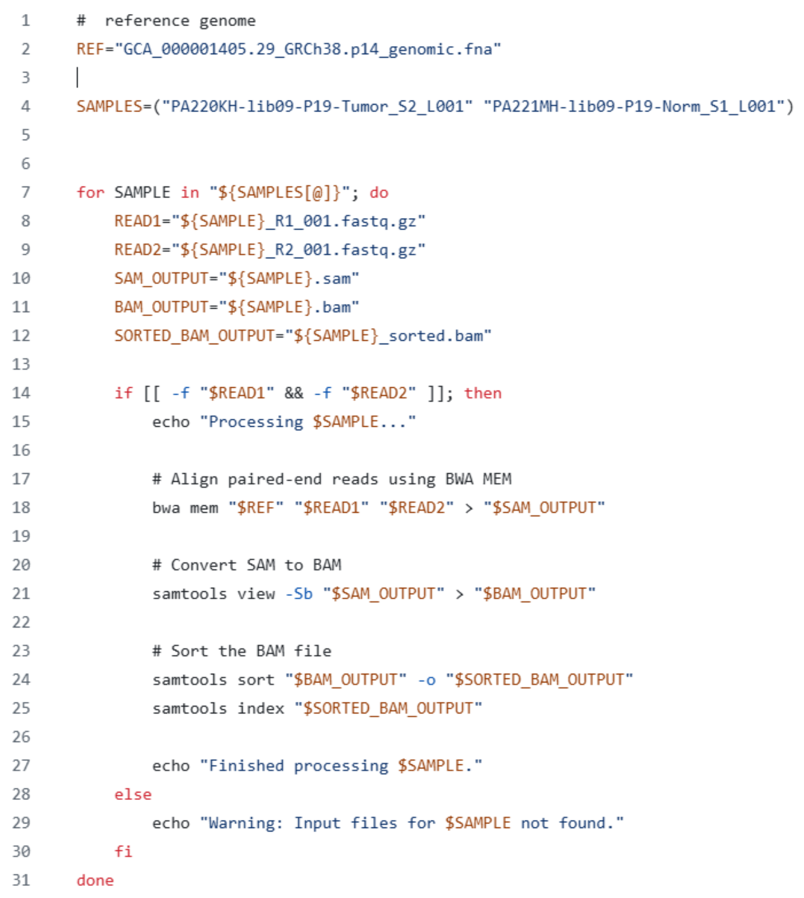


Fig 11: Samtools Flagstat: Read Count



Script-



```bash
#  reference genome
REF="GCA_000001405.29_GRCh38.p14_genomic.fna"

SAMPLES=("PA220KH-lib09-P19-Tumor_S2_L001" "PA221MH-lib09-P19-Norm_S1_L001")


for SAMPLE in "${SAMPLES[@]}"; do
    READ1="${SAMPLE}_R1_001.fastq.gz"
    READ2="${SAMPLE}_R2_001.fastq.gz"
    SAM_OUTPUT="${SAMPLE}.sam"
    BAM_OUTPUT="${SAMPLE}.bam"
    SORTED_BAM_OUTPUT="${SAMPLE}_sorted.bam"

    if [[ -f "$READ1" && -f "$READ2" ]]; then
        echo "Processing $SAMPLE..."

        # Align paired-end reads using BWA MEM
        bwa mem "$REF" "$READ1" "$READ2" > "$SAM_OUTPUT"

        # Convert SAM to BAM
        samtools view -Sb "$SAM_OUTPUT" > "$BAM_OUTPUT"

        # Sort the BAM file
        samtools sort "$BAM_OUTPUT" -o "$SORTED_BAM_OUTPUT"
        samtools index "$SORTED_BAM_OUTPUT"

        echo "Finished processing $SAMPLE."
    else
        echo "Warning: Input files for $SAMPLE not found."
    fi
done
```

Result –

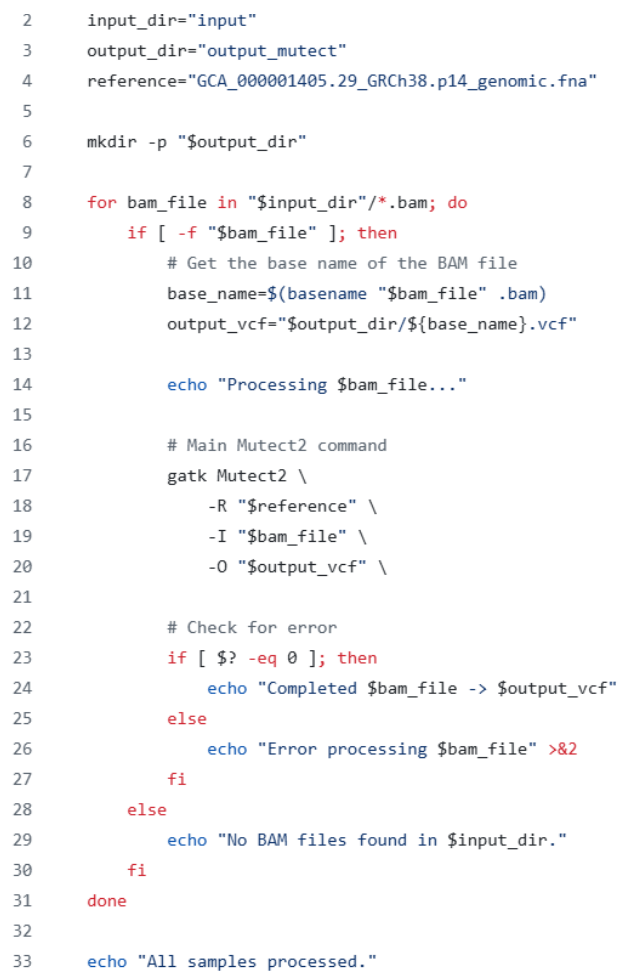
Alignment score is more then 99% for each sample

b. Identify somatic mutations present in the cancer sample but absent in the normal tissue.

i. Benchmark Software: Use established tools such as Mutect2, Strelka2, or VarScan2 for somatic mutation identification and background mutation estimation.

Mutect2

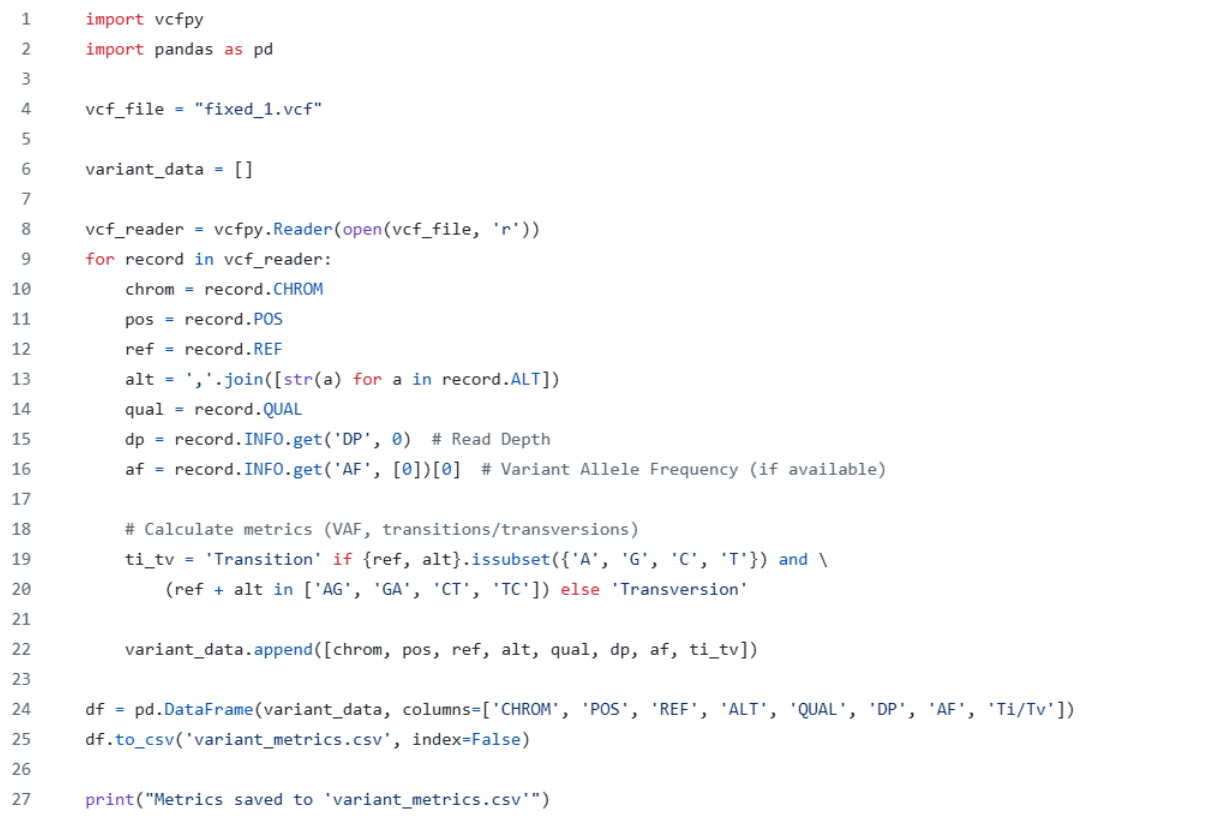
Codes-



```bash
input_dir="input"
output_dir="output_mutect"
reference="GCA_000001405.29_GRCh38.p14_genomic.fna"

mkdir -p "$output_dir"

for bam_file in "$input_dir"/*.bam; do
    if [ -f "$bam_file" ]; then
        # Get the base name of the BAM file
        base_name=$(basename "$bam_file" .bam)
        output_vcf="$output_dir/${base_name}.vcf"

        echo "Processing $bam_file..."

        # Main Mutect2 command
        gatk Mutect2 \
            -R "$reference" \
            -I "$bam_file" \
            -O "$output_vcf" \

        # Check for error
        if [ $? -eq 0 ]; then
            echo "Completed $bam_file -> $output_vcf"
        else
            echo "Error processing $bam_file" >&2
        fi
    else
        echo "No BAM files found in $input_dir."
    fi
done

echo "All samples processed."
```

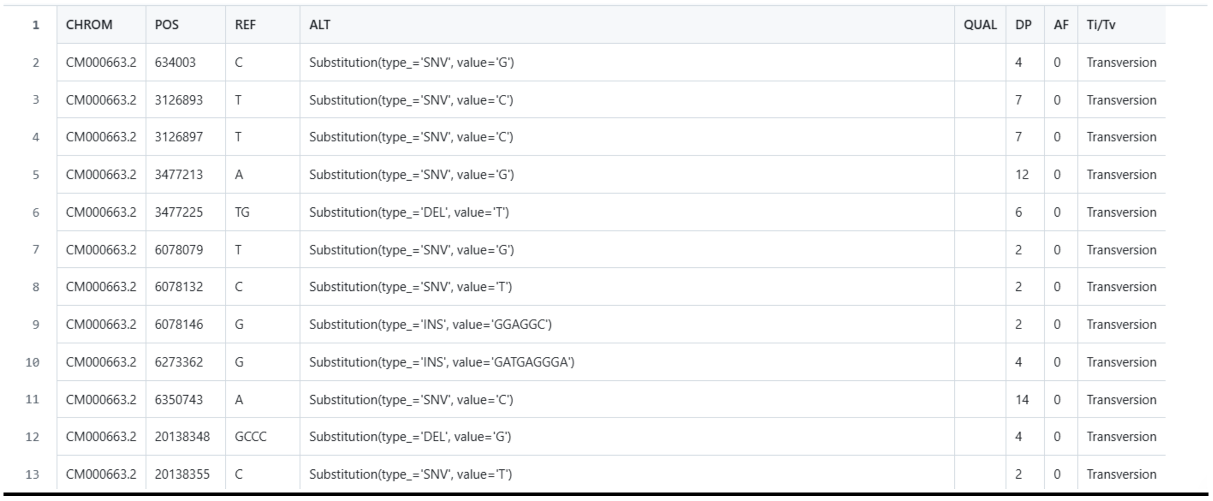
ii. **Custom Code Development:** Write your own scripts, leveraging tools like Samtools, bcftools, or Python/R libraries, to perform mutation detection and calculate the required metrics

<u>code-</u>



```python
import vcfpy
import pandas as pd

vcf_file = "fixed_1.vcf"

variant_data = []

vcf_reader = vcfpy.Reader(open(vcf_file, 'r'))
for record in vcf_reader:
    chrom = record.CHROM
    pos = record.POS
    ref = record.REF
    alt = ','.join([str(a) for a in record.ALT])
    qual = record.QUAL
    dp = record.INFO.get('DP', 0)  # Read Depth
    af = record.INFO.get('AF', [0])[0]  # Variant Allele Frequency (if available)

    # Calculate metrics (VAF, transitions/transversions)
    ti_tv = 'Transition' if {ref, alt}.issubset({'A', 'G', 'C', 'T'}) and \
        (ref + alt in ['AG', 'GA', 'CT', 'TC']) else 'Transversion'

    variant_data.append([chrom, pos, ref, alt, qual, dp, af, ti_tv])

df = pd.DataFrame(variant_data, columns=['CHROM', 'POS', 'REF', 'ALT', 'QUAL', 'DP', 'AF', 'Ti/Tv'])
df.to_csv('variant_metrics.csv', index=False)

print("Metrics saved to 'variant_metrics.csv'")
```

<u>Result(sample)-</u>



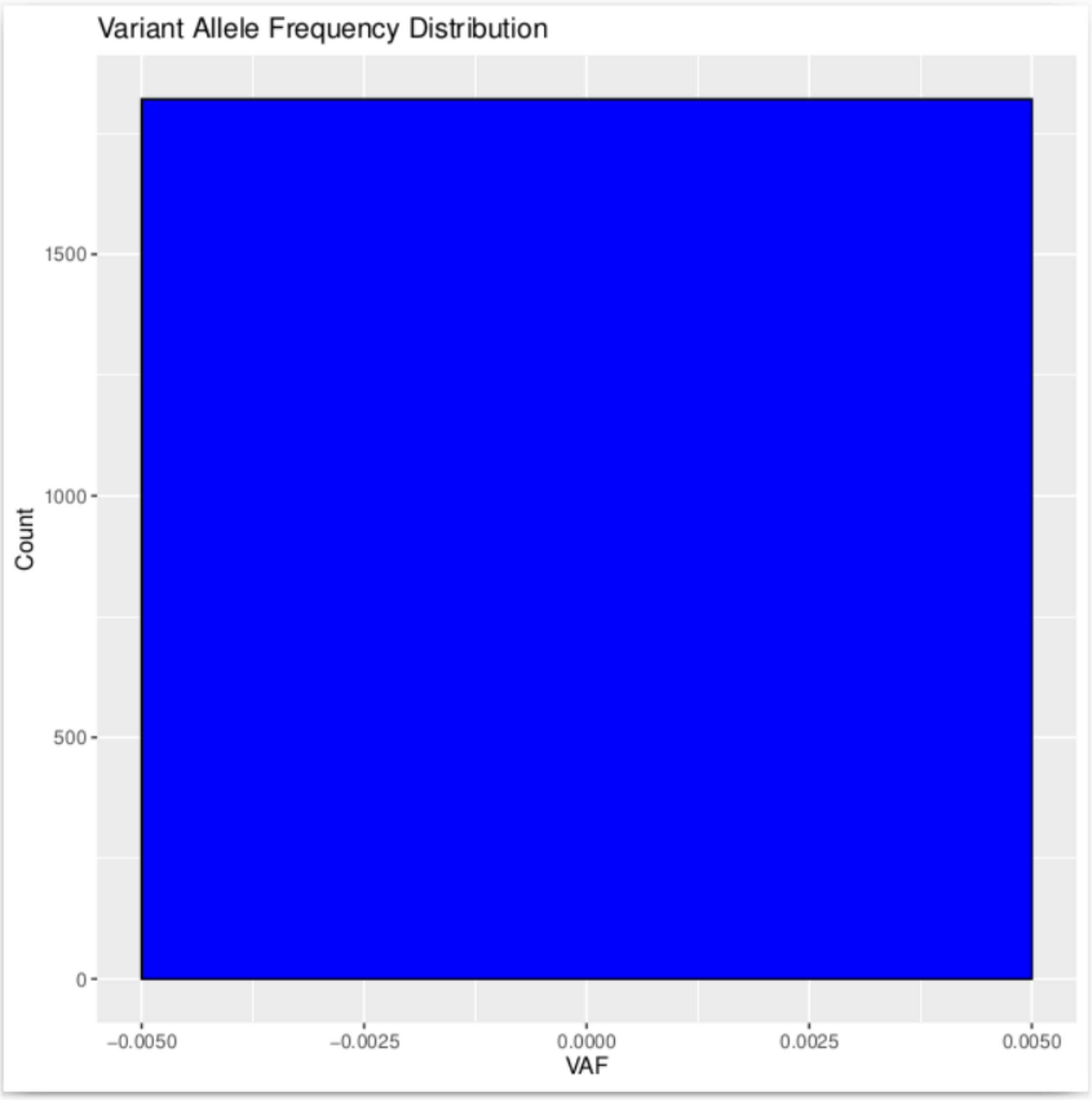
CHROM	POS	REF	ALT	QUAL	DP	AF	Ti/Tv
CM000663.2	634003	C	Substitution(type_='SNV', value='G')		4	0	Transversion
CM000663.2	3126893	T	Substitution(type_='SNV', value='C')		7	0	Transversion
CM000663.2	3126897	T	Substitution(type_='SNV', value='C')		7	0	Transversion
CM000663.2	3477213	A	Substitution(type_='SNV', value='G')		12	0	Transversion
CM000663.2	3477225	TG	Substitution(type_='DEL', value='T')		6	0	Transversion
CM000663.2	6078079	T	Substitution(type_='SNV', value='G')		2	0	Transversion
CM000663.2	6078132	C	Substitution(type_='SNV', value='T')		2	0	Transversion
CM000663.2	6078146	G	Substitution(type_='INS', value='GGAGGC')		2	0	Transversion
CM000663.2	6273362	G	Substitution(type_='INS', value='GATGAGGGA')		4	0	Transversion
CM000663.2	6350743	A	Substitution(type_='SNV', value='C')		14	0	Transversion
CM000663.2	20138348	GCCC	Substitution(type_='DEL', value='G')		4	0	Transversion
CM000663.2	20138355	C	Substitution(type_='SNV', value='T')		2	0	Transversion

Visualisation-

Code-

```r
# load package
library(ggplot2)

variant_data <- read.csv("variant_metrics.csv")
# Calculate overall Ti/Tv ratio
ti_count <- sum(variant_data$Ti.Tv == "Transition")
tv_count <- sum(variant_data$Ti.Tv == "Transversion")
ti_tv_ratio <- ti_count / tv_count
cat("Transition/Transversion Ratio:", ti_tv_ratio, "\n")

# Plot VAF distribution
ggplot(variant_data, aes(x = AF)) +
  geom_histogram(binwidth = 0.01, fill = "blue", color = "black") +
  labs(title = "Variant Allele Frequency Distribution", x = "VAF", y = "Count")
```

Result-



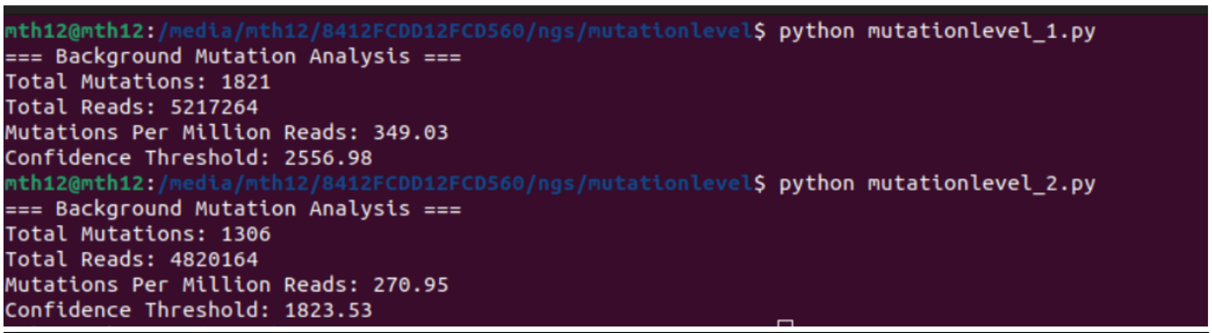
c. Use the normal tissue to calculate the median background mutation level. The background mutation level accounts for sequencing errors or biases that can mimic true mutations. Determine how many reads per million are required to confidently call a given mutation.

Code-

```python
import numpy as np
import pysam
def calculate_background_mutation_rate(vcf_file, bam_file):
    """
    Calculate the median background mutation rate and required reads for confident mutation calls.
    Parameters:
    - vcf_file (str): fixed_1.vcf
    - bam_file (str): fixed_1.bam
    Returns:
    - dict: Median mutation rate, total mutations, total reads, mutations per million reads.
    """
    # Count variants from the VCF file
    with open(vcf_file, 'r') as vcf:
        variants = [line for line in vcf if not line.startswith("#")]
    total_mutations = len(variants)
    # Calculate total reads from the BAM file
    bam = pysam.AlignmentFile(bam_file, "rb")
    total_reads = sum([stats.mapped for stats in bam.get_index_statistics()])
    bam.close()
    #  Calculate mutations per million reads
    mutations_per_million_reads = (total_mutations / total_reads) * 1e6
    # Calculate confidence threshold (optional scaling for higher confidence)
    confidence_threshold = mutations_per_million_reads + 1.5 * (np.median(np.abs(total_mutations - mutations_per_million_reads)))
    return {
        "Total Mutations": total_mutations,
        "Total Reads": total_reads,
        "Mutations Per Million Reads": mutations_per_million_reads,
        "Confidence Threshold": confidence_threshold,
    }
vcf_file = "fixed_1.vcf"
bam_file = "fixed_1.bam"
```

```python
results = calculate_background_mutation_rate(vcf_file, bam_file)
print("=== Background Mutation Analysis ===")
print(f"Total Mutations: {results['Total Mutations']}")
print(f"Total Reads: {results['Total Reads']}")
print(f"Mutations Per Million Reads: {results['Mutations Per Million Reads']:.2f}")
print(f"Confidence Threshold: {results['Confidence Threshold']:.2f}")
```

Result-



```
mth12@mth12:/media/mth12/8412FCDD12FCD560/ngs/mutationlevel$ python mutationlevel_1.py
=== Background Mutation Analysis ===
Total Mutations: 1821
Total Reads: 5217264
Mutations Per Million Reads: 349.03
Confidence Threshold: 2556.98
mth12@mth12:/media/mth12/8412FCDD12FCD560/ngs/mutationlevel$ python mutationlevel_2.py
=== Background Mutation Analysis ===
Total Mutations: 1306
Total Reads: 4820164
Mutations Per Million Reads: 270.95
Confidence Threshold: 1823.53
```