

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-

```

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')

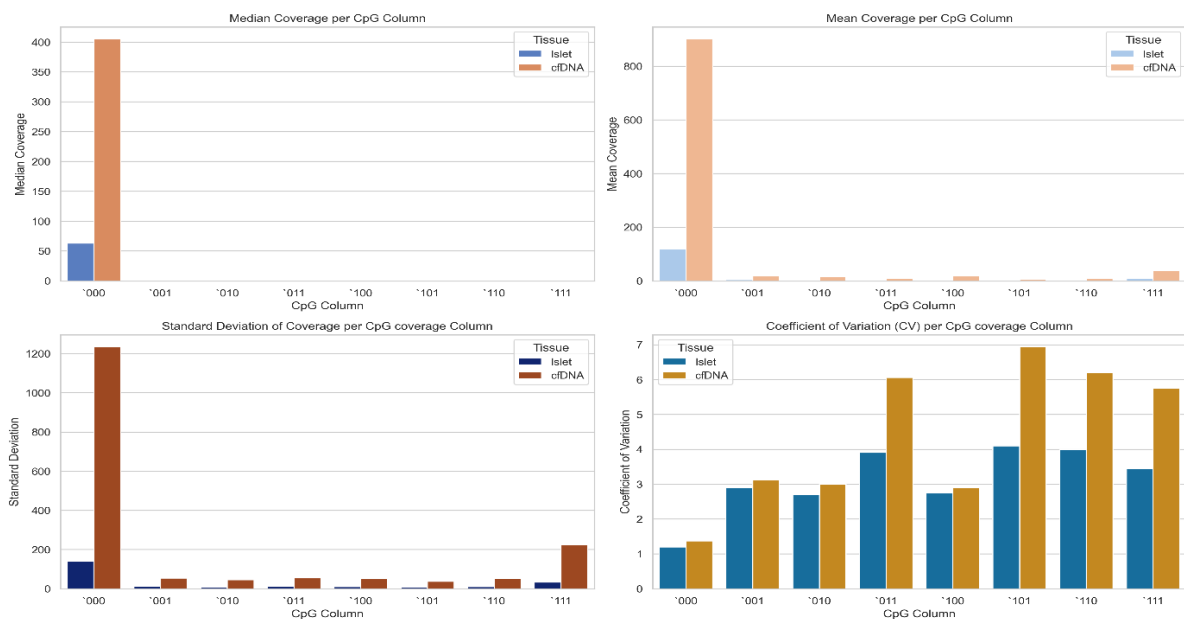
# 3. Standard Deviation plot
plt.subplot(2, 2, 3)
sns.barplot(data=results, x='CpG_Column', y='StdDev', hue='Tissue', palette='dark')
plt.title('Standard Deviation of Coverage per CpG coverage Column')
plt.xlabel('CpG Column')
plt.ylabel('Standard Deviation')

# 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

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-

```
1 import pandas as pd
2 from scipy.stats import fisher_exact
3 from statsmodels.stats.multitest import multipletests
4 chunk_size = 500
5 file_path = "PupilBioTest.csv"
6 methylation_cols = ['000', '001', '010', '011', '100', '101', '110', '111']
7
8 # initialize an empty list to store melted chunks
9 melted_chunks = []
10
11 # process the dataset in chunks
12 for chunk in pd.read_csv(file_path, chunksize=chunk_size):
13
14     melted_chunk = chunk.melt(
15         id_vars=['strand', 'CpG_Coordinates', 'Sample_ID', 'Replicate', 'Tissue'],
16         value_vars=methylation_cols,
17         var_name='Methylation_Status',
18         value_name='Count'
19     )
20     melted_chunks.append(melted_chunk)
21
22 # combine into a single DataFrame
23 df_melted = pd.concat(melted_chunks, ignore_index=True)
24
25 # group data by tissue, strand, coordinates, and methylation status
26 grouped = df_melted.groupby(['Tissue', 'strand', 'CpG_Coordinates', 'Methylation_Status']).agg(
27     Total_Count=('Count', 'sum')
28 ).reset_index()
29
30 # Normalize counts by total counts per tissue
31 grouped['Normalized_Count'] = grouped['Total_Count'] / grouped.groupby('Tissue')['Total_Count'].transform('sum')
32
33 # comparison table
34 tissue1 = 'cfdNA'
35 tissue2 = 'islet'
36
37 tissue1_data = grouped[grouped['Tissue'] == tissue1]
38 tissue2_data = grouped[grouped['Tissue'] == tissue2]
39
40 comparison = pd.merge(
41     tissue1_data[['strand', 'CpG_Coordinates', 'Methylation_Status', 'Total_Count']],
42     tissue2_data[['strand', 'CpG_Coordinates', 'Methylation_Status', 'Total_Count']],
43     on=['strand', 'CpG_Coordinates', 'Methylation_Status'],
44     how='outer',
45     suffixes=('_T1', '_T2')
46 ).fillna(0)
47
48 # Fisher's Exact Test for each PMP
49 p_values = []
50 for _, row in comparison.iterrows():
51     contingency_table = [
```

```

51         [row['Total_Count_T1'], sum(tissue1_data['Total_Count']) - row['Total_Count_T1']],
52         [row['Total_Count_T2'], sum(tissue2_data['Total_Count']) - row['Total_Count_T2']],
53     ]
54     _, p = fisher_exact(contingency_table)
55     p_values.append(p)
56
57     comparison['P_Value'] = p_values
58
59     # adusted p-values for multiple testing with error handling
60     if not comparison['P_Value'].isnull().all() and len(comparison['P_Value']) > 0:
61         # Remove NaN P-values and apply multiple testing correction
62         valid_p_values = comparison['P_Value'].dropna()
63         if len(valid_p_values) > 0:
64             try:
65                 comparison['Adjusted_P_Value'] = multipletests(valid_p_values, method='fdr_bh')[1]
66             except Exception as e:
67                 print(f"Error during multiple testing correction: {e}")
68                 comparison['Adjusted_P_Value'] = None
69         else:
70             print("Error: No valid P-values found for multiple testing correction.")
71             comparison['Adjusted_P_Value'] = None
72     else:
73         print("Error: No valid P-values found for multiple testing correction.")
74         comparison['Adjusted_P_Value'] = None
75
76     # Filter PMPs
77     significant_pmps = comparison[comparison['Adjusted_P_Value'] < 0.05]
78
79     significant_pmps.to_csv('statistical_significant_pmps.csv', index=False)
80     print("Significant PMPs based on statistical approach saved to 'statistical_significant_pmps.csv'")

```

Result (sample) –

	strand	CpG_Coordinates	Methylation_Status	Total_Count_T1	Total_Count_T2	P_Value	Adjusted_P_Value
1							
2	r	10035:10044:10046	'000	1077.0	0.0	1.3382072844487038e-36	1.7676116401349276e-35
3	r	10035:10044:10053	'000	1086.0	0.0	5.917675284589969e-37	8.023259140671622e-36
4	r	10035:10044:10058	'000	1069.0	0.0	1.908798556018049e-36	2.497462623036202e-35
5	r	10035:10044:10071	'000	1072.0	0.0	1.9906808891810815e-36	2.5918875831200024e-35
6	r	10035:10044:10075	'000	1086.0	0.0	5.917675284589969e-37	8.023259140671622e-36
7	r	10035:10044:10081	'000	1086.0	0.0	5.917675284589969e-37	8.023259140671622e-36
8	r	10035:10044:10088	'000	1047.0	0.0	1.452516828653845e-35	1.810716014159768e-34
9	r	10035:10044:10096	'000	1056.0	0.0	6.371907791190341e-36	8.078305354230393e-35
10	r	10035:10044:10102	'000	1054.0	0.0	6.226306510017574e-36	7.951061266139723e-35
11	r	10035:10044:10109	'000	1005.0	0.0	3.281449035350466e-34	3.921801741714632e-33
12	r	10035:10044:10111	'000	1055.0	0.0	6.263405682991248e-36	7.969487437799825e-35
13	r	10035:10044:10118	'000	955.0	0.0	1.7386811904872064e-32	2.019134815267068e-31
14	r	10035:10044:10121	'000	907.0	0.0	5.7755991783838745e-31	6.59517144230489e-30
15	r	10035:10044:10130	'000	797.0	0.0	3.550894010064119e-27	3.7663965430886996e-26
16	r	10035:10044:10133	'000	764.0	0.0	3.604335951332221e-26	3.7630674200417176e-25
17	r	10035:10044:10140	'000	620.0	0.0	2.2639107299259707e-21	2.165189088302304e-20
18	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 -

```
1 import pandas as pd
2 file_path = "PupilBioTest.csv" # Replace with your actual file name
3 methylation_cols = ['000', '001', '010', '011', '100', '101', '110', '111']
4 df = pd.read_csv(file_path)
5 df.columns = df.columns.str.strip()
6 required_columns = ['strand', 'CpG_Coordinates', 'Sample_ID', 'Replicate', 'Tissue'] + methylation_cols
7 for col in required_columns:
8     if col not in df.columns:
9         raise ValueError(f"Column '{col}' not found in the dataset. Please check your input file.")
10 # melt the dataset to create a 'Methylation_Status' column
11 df_melted = df.melt(
12     id_vars=['strand', 'CpG_Coordinates', 'Sample_ID', 'Replicate', 'Tissue'],
13     value_vars=methylation_cols,
14     var_name='Methylation_Status',
15     value_name='Count'
16 )
17 # Create PMP
18 df_melted['PMP'] = [
19     df_melted['strand'] + ':' + df_melted['CpG_Coordinates'] + ':' + df_melted['Methylation_Status']
20 ]
21 # group data by Tissue and PMP to calculate total counts
22 grouped = df_melted.groupby(['Tissue', 'PMP']).agg(
23     Total_Count=('Count', 'sum')
24 ).reset_index()
25 # Calculate the total counts for each tissue
26 grouped['Total_Tissue_Count'] = grouped.groupby('Tissue')['Total_Count'].transform('sum')
27 # Calculate VRF
28 grouped['VRF'] = grouped['Total_Count'] / grouped['Total_Tissue_Count']
29 output_file = "vrf_results.csv"
30 grouped.to_csv(output_file, index=False)
31 print(f"VRF results saved to {output_file}")
```

Result –

	A	B	C	D	E	F
1	Tissue	PMP	Total_Count	Total_Tissue_Count	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 Sensitivity



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-

```
1 import pandas as pd
2
3 file_path = "statistical_significant_pmps.csv" # Use the provided data file
4 data = pd.read_csv(file_path)
5
6 # sort by Adjusted P-Value (column 'Adjusted_P_Value') to get the top 10 PMPs
7 top_pmps = data.sort_values(by='Adjusted_P_Value').head(10)
8
9 # assuming a sequencing depth of 1 million reads for Tissue #2
10 sequencing_depth_t2 = 1_000_000
11
12 # estimate the threshold of reads required for each PMP
13 top_pmps['Threshold_Reads_T2'] = top_pmps['Total_Count_T2'] / top_pmps['Total_Count_T2'].sum() * sequencing_depth_t2
14
15 print(top_pmps[['CpG_Coordinates', 'Methylation_Status', 'Total_Count_T2', 'Threshold_Reads_T2']])
```

Result –

```
mp_results_saved_to_VIP_results.csv
mth12@mth12:~/downloads/pup11_bio/blomarker_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-

```
1  import pandas as pd
2  from scipy.stats import fisher_exact
3  from statsmodels.stats.multitest import multipletests
4  import matplotlib.pyplot as plt
5
6  file_path = "PupilBioTest.csv"
7
8  methylation_cols = ['000', '001', '010', '011', '100', '101', '110', '111']
9
10 # Initialize result storage
11 pmp_results = []
12 cp_g_results = []
13
14 # Process data in chunks
15 chunk_size = 200
16 for chunk in pd.read_csv(file_path, chunksize=chunk_size):
17     melted_chunk = chunk.melt(
18         id_vars=['strand', 'CpG_Coordinates', 'Sample_ID', 'Replicate', 'Tissue'],
19         value_vars=methylation_cols,
20         var_name='Methylation_Status',
21         value_name='Count'
22     )
23
24     grouped = melted_chunk.groupby(['Tissue', 'strand', 'CpG_Coordinates', 'Methylation_Status']).agg(
25         Total_Count=('Count', 'sum')
26     ).reset_index()
27
28     tissue1 = 'cDNA'
29     tissue2 = 'Islet'
30
31     tissue1_data = grouped[grouped['Tissue'] == tissue1]
32     tissue2_data = grouped[grouped['Tissue'] == tissue2]
33
34     # PMPs Analysis
35     comparison = pd.merge(
36         tissue1_data[['strand', 'CpG_Coordinates', 'Methylation_Status', 'Total_Count']],
37         tissue2_data[['strand', 'CpG_Coordinates', 'Methylation_Status', 'Total_Count']],
38         on=['strand', 'CpG_Coordinates', 'Methylation_Status'],
39         how='outer',
40         suffixes=('_T1', '_T2')
41     ).fillna(0)
42
43     for _, row in comparison.iterrows():
44         contingency_table = [
45             [int(row['Total_Count_T1']), int(sum(tissue1_data['Total_Count']) - row['Total_Count_T1'])],
46             [int(row['Total_Count_T2']), int(sum(tissue2_data['Total_Count']) - row['Total_Count_T2'])],
47         ]
48         if any(x < 0 for contingency_row in contingency_table for x in contingency_row) or sum(sum(contingency_table, [])) == 0:
49             continue
50
51         _, p_value = fisher_exact(contingency_table)
52
53         true_negatives = contingency_table[1][1]
54         total_negatives = contingency_table[1][0] + contingency_table[1][1]
55         specificity = true_negatives / total_negatives if total_negatives > 0 else 0
56
57         pmp_results.append((row['CpG_Coordinates'], row['Methylation_Status'], p_value, specificity))
58
59     # CpG Site Analysis
60     individual_cp_g = melted_chunk.groupby(['CpG_Coordinates', 'Tissue']).agg(
61         Total_Count=('Count', 'sum')
62     ).reset_index()
63
64     for cp_g, group in individual_cp_g.groupby('CpG_Coordinates'):
65         tissue_counts = group.set_index('Tissue')['Total_Count'].to_dict()
```

```

66         count_t1 = tissue_counts.get(tissue1, 0)
67         count_t2 = tissue_counts.get(tissue2, 0)
68         contingency_table = [
69             [count_t1, sum(group['Total_Count']) - count_t1],
70             [count_t2, sum(group['Total_Count']) - count_t2],
71         ]
72         if any(x < 0 for contingency_row in contingency_table for x in contingency_row) or sum(sum(contingency_table, [])) == 0:
73             continue
74
75         _, p_value = fisher_exact(contingency_table)
76
77         true_negatives = contingency_table[1][1]
78         total_negatives = contingency_table[1][0] + contingency_table[1][1]
79         specificity = true_negatives / total_negatives if total_negatives > 0 else 0
80         cpg_results.append((cpg, p_value, specificity))
81
82     # Convert results to DataFrame
83     pmp_df = pd.DataFrame(pmp_results, columns=['CpG_Coordinates', 'Methylation_Status', 'P_Value', 'Specificity'])
84     cpg_df = pd.DataFrame(cpg_results, columns=['CpG_Coordinates', 'P_Value', 'Specificity'])
85
86     # Adjust p-values
87     pmp_df['Adjusted_P_Value'] = multipletests(pmp_df['P_Value'], method='fdr_bh')[1]
88     cpg_df['Adjusted_P_Value'] = multipletests(cpg_df['P_Value'], method='fdr_bh')[1]
89
90     # Filter top 10 PMPs and CpG sites
91     top_pmps = pmp_df.sort_values(by='Specificity', ascending=False).head(10)
92     top_cpgs = cpg_df.sort_values(by='Specificity', ascending=False).head(10)
93
94     # Compare Specificity Distributions
95     plt.boxplot([top_pmps['Specificity'], top_cpgs['Specificity']], labels=['PMPs', 'CpG Sites'])
96     plt.ylabel('Specificity')
97     plt.title('Specificity Comparison: PMPs vs CpG Sites')
98     plt.show()
99
100    top_pmps.to_csv("top_10_pmps.csv", index=False)
101    top_cpgs.to_csv("top_10_cpg_sites.csv", index=False)
102    print("Top 10 PMPs and CpG sites saved. Specificity comparison completed.")

```

Result –

Top 10 cpg sites

1	CpG_Coordinates	P_Value	Specificity	Adjusted_P_Value
2	10035:10044:10046	0.0	1.0	0.0
3	13627:13636:13707	0.0	1.0	0.0
4	13627:13724:13742	0.0	1.0	0.0
5	13627:13724:13760	0.0	1.0	0.0
6	13627:13724:13771	1.852955281220998e-131	1.0	2.3630557400645897e-131
7	13627:13742:13760	0.0	1.0	0.0
8	13627:13742:13771	1.1633432636587346e-132	1.0	1.4860647338099116e-132
9	13627:13760:13771	4.642892484692069e-132	1.0	5.923491199880882e-132
10	13634:13636:13647	0.0	1.0	0.0
11	13634:13636:13661	0.0	1.0	0.0

Top 10 PMP'S

1	CpG_Coordinates	Methylation_Status	P_Value	Specificity	Adjusted_P_Value
2	10035:10044:10046	`000	1.0	0	1.0
3	13636:13647:13678	`110	1.0	0	1.0
4	13636:13647:13678	`100	1.0	0	1.0
5	13636:13647:13678	`011	1.0	0	1.0
6	13636:13647:13678	`010	1.0	0	1.0
7	13636:13647:13678	`001	1.0	0	1.0
8	13636:13647:13678	`000	1.0	0	1.0
9	13636:13647:13669	`111	1.0	0	1.0
10	13636:13647:13669	`110	1.0	0	1.0
11	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.

Sub-tasks:

1. Quality Control:

- a. Perform quality checks using tools like FastQC and summarize quality metrics (e.g., sequence counts, per-base quality, read duplication levels).

command to run fastqc

fastqc <filename.fastq> -o <output_directory>

2. Alignment and Mutation Calling

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

Code-

```
1  # reference genome
2  REF="GCA_000001405.29_GRCh38.p14_genomic.fna"
3  |
4  SAMPLES=("PA220KH-lib09-P19-Tumor_S2_L001" "PA221MH-lib09-P19-Norm_S1_L001")
5
6
7  for SAMPLE in "${SAMPLES[@]"; do
8      READ1="${SAMPLE}_R1_001.fastq.gz"
9      READ2="${SAMPLE}_R2_001.fastq.gz"
10     SAM_OUTPUT="${SAMPLE}.sam"
11     BAM_OUTPUT="${SAMPLE}.bam"
12     SORTED_BAM_OUTPUT="${SAMPLE}_sorted.bam"
13
14     if [[ -f "$READ1" && -f "$READ2" ]]; then
15         echo "Processing $SAMPLE..."
16
17         # Align paired-end reads using BWA MEM
18         bwa mem "$REF" "$READ1" "$READ2" > "$SAM_OUTPUT"
19
20         # Convert SAM to BAM
21         samtools view -Sb "$SAM_OUTPUT" > "$BAM_OUTPUT"
22
23         # Sort the BAM file
24         samtools sort "$BAM_OUTPUT" -o "$SORTED_BAM_OUTPUT"
25         samtools index "$SORTED_BAM_OUTPUT"
26
27         echo "Finished processing $SAMPLE."
28     else
29         echo "Warning: Input files for $SAMPLE not found."
30     fi
31 done
```

Result –

Alignment score is more than 95% 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-

```
2   input_dir="input"
3   output_dir="output_mutect"
4   reference="GCA_000001405.29_GRCh38.p14_genomic.fna"
5
6   mkdir -p "$output_dir"
7
8   for bam_file in "$input_dir"/*.bam; do
9       if [ -f "$bam_file" ]; then
10          # Get the base name of the BAM file
11          base_name=$(basename "$bam_file" .bam)
12          output_vcf="$output_dir/${base_name}.vcf"
13
14          echo "Processing $bam_file..."
15
16          # Main Mutect2 command
17          gatk Mutect2 \
18              -R "$reference" \
19              -I "$bam_file" \
20              -O "$output_vcf" \
21
22          # Check for error
23          if [ $? -eq 0 ]; then
24              echo "Completed $bam_file -> $output_vcf"
25          else
26              echo "Error processing $bam_file" >&2
27          fi
28      else
29          echo "No BAM files found in $input_dir."
30      fi
31  done
32
33  echo "All samples processed."
```

Results –

VCF format files

PA220KH-lib09-P19-Tumor_S1_L001_R1_001.fastq.gz

PA220KH-lib09-P19-Tumor_S1_L001_R2_001.fastq.gz

PA220KH-lib09-P19-Tumor_S2_L001_R1_001.fastq.gz

PA220KH-lib09-P19-Tumor_S2_L001_R2_001.fastq.gz



Fixed_1.vcf



Fixed_2.vcf

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

code-

```

1  import vcfpy
2  import pandas as pd
3
4  vcf_file = "fixed_1.vcf"
5
6  variant_data = []
7
8  vcf_reader = vcfpy.Reader(open(vcf_file, 'r'))
9  for record in vcf_reader:
10     chrom = record.CHROM
11     pos = record.POS
12     ref = record.REF
13     alt = ','.join([str(a) for a in record.ALT])
14     qual = record.QUAL
15     dp = record.INFO.get('DP', 0) # Read Depth
16     af = record.INFO.get('AF', [0])[0] # Variant Allele Frequency (if available)
17
18     # Calculate metrics (VAF, transitions/transversions)
19     ti_tv = 'Transition' if {ref, alt}.issubset({'A', 'G', 'C', 'T'}) and \
20         (ref + alt in ['AG', 'GA', 'CT', 'TC']) else 'Transversion'
21
22     variant_data.append([chrom, pos, ref, alt, qual, dp, af, ti_tv])
23
24 df = pd.DataFrame(variant_data, columns=['CHROM', 'POS', 'REF', 'ALT', 'QUAL', 'DP', 'AF', 'Ti/Tv'])
25 df.to_csv('variant_metrics.csv', index=False)
26
27 print("Metrics saved to 'variant_metrics.csv'")

```

Result(sample)-

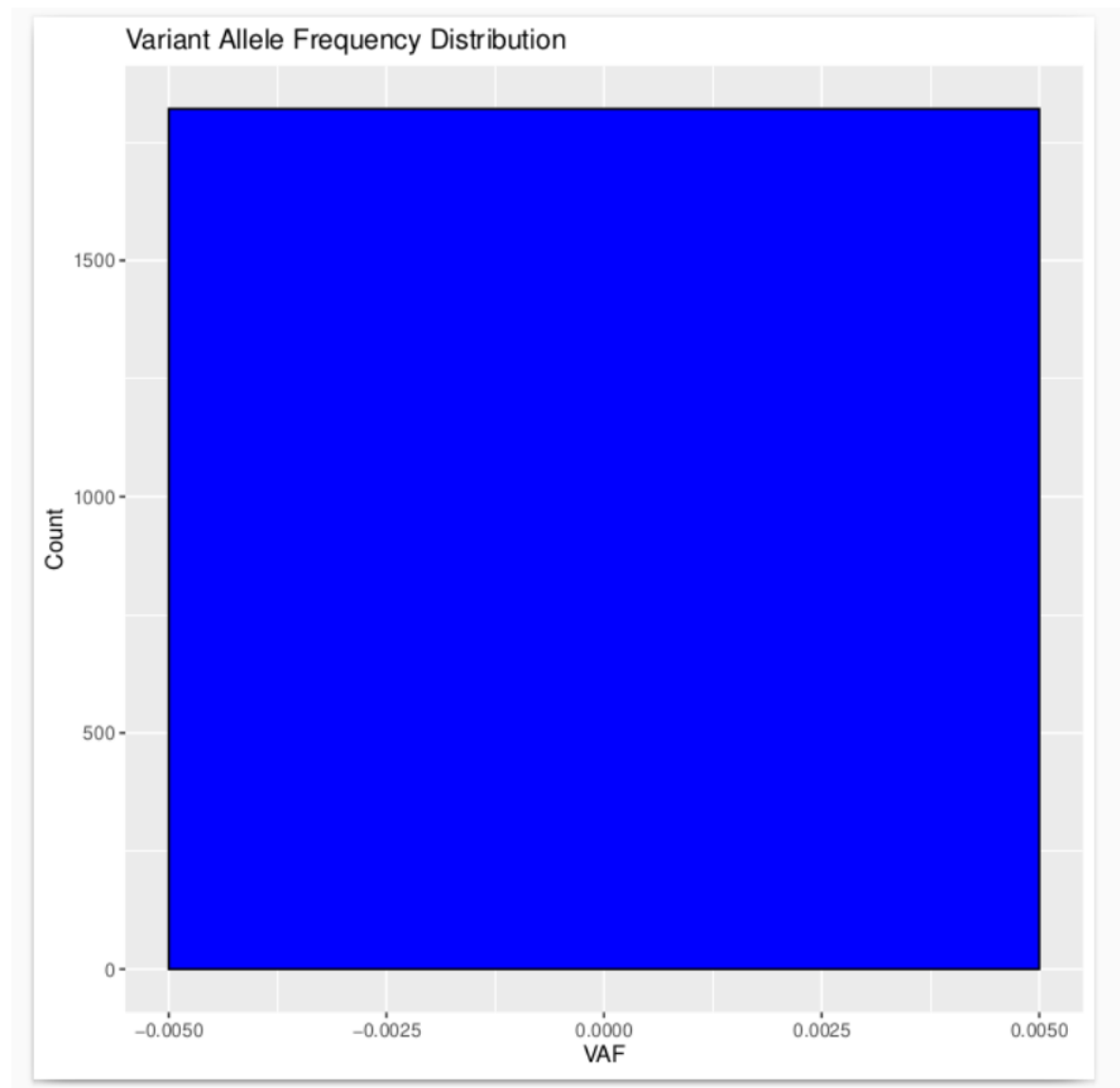
	CHROM	POS	REF	ALT	QUAL	DP	AF	Ti/Tv
1	CM000663.2	634003	C	Substitution(type='SNV', value='G')		4	0	Transversion
2	CM000663.2	3126893	T	Substitution(type='SNV', value='C')		7	0	Transversion
3	CM000663.2	3126897	T	Substitution(type='SNV', value='C')		7	0	Transversion
4	CM000663.2	3477213	A	Substitution(type='SNV',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

Visualisation-

Code-

```
1 # load package
2 library(ggplot2)
3
4 variant_data <- read.csv("variant_metrics.csv")
5 # Calculate overall Ti/Tv ratio
6 ti_count <- sum(variant_data$Ti.Tv == "Transition")
7 tv_count <- sum(variant_data$Ti.Tv == "Transversion")
8 ti_tv_ratio <- ti_count / tv_count
9 cat("Transition/Transversion Ratio:", ti_tv_ratio, "\n")
10
11 # Plot VAF distribution
12 ggplot(variant_data, aes(x = AF)) +
13   geom_histogram(binwidth = 0.01, fill = "blue", color = "black") +
14   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-

```
1  import numpy as np
2  import pysam
3  def calculate_background_mutation_rate(vcf_file, bam_file):
4      """
5      Calculate the median background mutation rate and required reads for confident mutation calls.
6      Parameters:
7      - vcf_file (str): fixed_1.vcf
8      - bam_file (str): fixed_1.bam
9      Returns:
10     - dict: Median mutation rate, total mutations, total reads, mutations per million reads.
11     """
12     # Count variants from the VCF file
13     with open(vcf_file, 'r') as vcf:
14         variants = [line for line in vcf if not line.startswith("#")]
15     total_mutations = len(variants)
16     # Calculate total reads from the BAM file
17     bam = pysam.AlignmentFile(bam_file, "rb")
18     total_reads = sum([stats.mapped for stats in bam.get_index_statistics()])
19     bam.close()
20     # Calculate mutations per million reads
21     mutations_per_million_reads = (total_mutations / total_reads) * 1e6
22     # Calculate confidence threshold (optional scaling for higher confidence)
23     confidence_threshold = mutations_per_million_reads + 1.5 * (np.median(np.abs(total_mutations - mutations_per_million_reads)))
24     return {
25         "Total Mutations": total_mutations,
26         "Total Reads": total_reads,
27         "Mutations Per Million Reads": mutations_per_million_reads,
28         "Confidence Threshold": confidence_threshold,
29     }
30     vcf_file = "fixed_1.vcf"
31     bam_file = "fixed_1.bam"

32     results = calculate_background_mutation_rate(vcf_file, bam_file)
33     print("=== Background Mutation Analysis ===")
34     print(f"Total Mutations: {results['Total Mutations']}")
35     print(f"Total Reads: {results['Total Reads']}")
36     print(f"Mutations Per Million Reads: {results['Mutations Per Million Reads']:.2f}")
37     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
```