

# FinalProjPlink

March 6, 2024

```
[ ]: #The plink command we intend to replicate  
    """  
    plink --vcf ~/1_284FinalProject/ps3_gwas.vcf.gz --linear --maf 0.05 --pheno ~/  
    ↪1_284FinalProject/ps3_gwas.phen --out ps3_gwas --allow-no-sex  
    """
```

```
[1]: #Import all of the necessary libraries  
import pandas as pd  
import numpy as np  
import gzip  
from io import StringIO  
from tqdm import tqdm  
import statsmodels.api as sm
```

## 1 Data Preprocessing

```
[ ]: #Function to parse the vcf file and save to a dataframe  
def parse_vcf(vcf_path):  
    """  
    Parses the VCF file to extract SNP information, with progress indication.  
  
    Parameters:  
    vcf_path (str): Path to the VCF file.  
  
    Returns:  
    pd.DataFrame: DataFrame with SNP information, one row per SNP.  
    """  
  
    # Determine if the file is compressed and choose the appropriate opener  
    if vcf_path.endswith('.gz'):  
        opener = gzip.open  
    else:  
        opener = open  
  
    # Read the file and filter out the header lines  
    with opener(vcf_path, 'rt') as f:
```

```

    # Use tqdm to show progress. Wrapping f in tqdm() will not give the
    ↪correct total line count,
    # so we use it to display progress without total count, or preprocess
    ↪for total line count if needed.
    lines = [l for l in tqdm(f, desc="Reading VCF")]
    data_lines = [l for l in lines if not l.startswith('##')]

    # Create a DataFrame from the filtered lines
    vcf_df = pd.read_csv(StringIO(''.join(data_lines)), delimiter='\t',
    ↪dtype={'#CHROM': str, 'POS': int, 'ID': str, 'REF': str, 'ALT': str, 'QUAL':
    ↪str, 'FILTER': str, 'INFO': str})
    vcf_df.rename(columns={'#CHROM': 'CHROM'}, inplace=True)
    return vcf_df

```

```

[ ]: #Parsing the vcf file ps3_gwas.vcf.gz
    """
    Note: this function can take quite a while to run,
    and can be skipped for now if following dataframes are imported
    """

    vcfd = parse_vcf("ps3_gwas.vcf.gz")

```

```

[ ]: #Saving vcfd as csv
    #vcfd.to_csv("vcfdout.csv")

```

```

[7]: #Columns from vcfd
columns = [
    'CHROM', 'POS', 'ID', 'REF', 'ALT', 'QUAL', 'FILTER', 'INFO', 'FORMAT',
    ↪'NA06984', 'NA06989', 'NA12878', 'NA18489',
    'NA18504', 'NA18511', 'NA18516', 'NA18523', 'NA18908', 'NA18910',
    ↪'NA18915', 'NA18934', 'NA11832', 'NA11894', 'NA11919',
    'NA11933', 'NA11995', 'NA12006', 'NA12044', 'NA12234', 'NA12272',
    ↪'NA12342', 'NA12347', 'NA12400', 'NA12760', 'NA11829',
    'NA12777', 'NA11831', 'NA12828', 'NA11843', 'NA12830', 'NA11881',
    ↪'NA12842', 'NA11893', 'NA12873', 'NA11918', 'NA11920',
    'NA11932', 'NA11994', 'NA12005', 'NA12889', 'NA18488', 'NA19095',
    ↪'NA18508', 'NA18510', 'NA18522', 'NA18864', 'NA18871',
    'NA18876', 'NA12776', 'NA12815', 'NA12827', 'NA12872', 'NA12043',
    ↪'NA12144', 'NA12156', 'NA19153', 'NA19160', 'NA12283',
    'NA19172', 'NA12341', 'NA19184', 'NA07037', 'NA19189', 'NA07051',
    ↪'NA07056', 'NA07347', 'NA19204', 'NA19209', 'NA18856',
    'NA18868', 'NA18870', 'NA19099', 'NA19222', 'NA19239', 'NA19223',
    ↪'NA19235', 'NA19247', 'NA18907', 'NA18933', 'NA19108',
    'NA19141', 'NA19146', 'NA19152', 'NA19171', 'NA19190', 'NA19210',
    ↪'NA19102', 'NA19107', 'NA19114', 'NA19119', 'NA19121',

```

```

'NA19138', 'NA12273', 'NA12348', 'NA12413', 'NA12716', 'NA12761',
↪ 'NA12778', 'NA12812', 'NA12829', 'NA12843', 'NA12155',
'NA12874', 'NA12249', 'NA12275', 'NA12282', 'NA12287', 'NA06985',
↪ 'NA12340', 'NA12383', 'NA12489', 'NA10851', 'NA11830',
'NA10847', 'NA11892', 'NA11931', 'NA11840', 'NA12004', 'NA12045',
↪ 'NA06994', 'NA07000', 'NA07048', 'NA18853', 'NA18916',
'NA18923', 'NA19096', 'NA19093', 'NA18505', 'NA19098', 'NA18517',
↪ 'NA12890', 'NA18499', 'NA18502', 'NA18507', 'NA18519',
'NA19116', 'NA19130', 'NA19147', 'NA19159', 'NA18858', 'NA18865',
↪ 'NA18877', 'NA12718', 'NA18909', 'NA12749', 'NA12751',
'NA19248', 'NA12763', 'NA12775', 'NA12814', 'NA18879', 'NA18881',
↪ 'NA19185', 'NA19197', 'NA19113', 'NA19200', 'NA19118',
'NA19236', 'NA19137', 'NA19144', 'NA19149', 'NA19175', 'NA19207',
↪ 'NA19214', 'NA18867', 'NA18874', 'NA19238', 'NA19257',
'NA07357', 'NA06986', 'NA18486', 'NA18498', 'NA18501', 'NA18520',
↪ 'NA19092', 'NA12154', 'NA12286', 'NA12399', 'NA12414',
'NA12546', 'NA12717', 'NA12748', 'NA12750', 'NA12762', 'NA12813',
↪ 'NA18912', 'NA18917', 'NA18924', 'NA11930', 'NA11992',

↪
↪ 'NA12003', 'NA12046', 'NA12058', 'NA18861', 'NA18873', 'NA18878', 'NA19256', 'NA19198', 'NA19201', 'NA19117', 'NA19129', 'NA19131', 'NA19143']

```

```

[ ]: # Function to parse genotypes and calculate allele counts
def allele_counts(row):
    alleles = row[9:].str.extractall(r'(\d)')[0] # Extracting alleles from
    ↪ genotypes, assuming genotype data starts from the 10th column
    allele_counts = alleles.value_counts()
    return allele_counts

# Calculate allele counts for each SNP
df_allele_counts = vcfd.f.apply(allele_counts, axis=1)

# Calculate MAF for each SNP
total_alleles = 2 * (len(vcfd.f.columns) - 9) # Total alleles = 2 * number of
    ↪ samples, assuming genotype data starts from the 10th column
vcfd.f['MAF'] = df_allele_counts.apply(lambda x: x.min() / total_alleles if not
    ↪ x.empty else np.nan, axis=1)

# Filter SNPs with MAF < 0.05
filtered_df = vcfd.f[vcfd.f['MAF'] >= 0.05]

# Save filtered_df to csv
filtered_df.to_csv("filtered_df.csv")

```

```

[ ]: #filtered_df = pd.read_csv("filtered_df.csv")

```

```
[ ]: #Dropping the first column from the DataFrame
filtered_df = filtered_df.drop(columns=[filtered_df.columns[0]])

[ ]: #Dropping columns that arent SNPs
filtered_df = filtered_df.drop(['CHROM', 'POS', 'REF', 'ALT', 'QUAL', 'FILTER', 'INFO', 'FORMAT'], axis=1)

[9]: #Reading in phenotype information from ps3_gwas.phen
phenotype_df = pd.read_csv('ps3_gwas.phen', sep='\t', header=None, names=['SampleID', 'PhenotypeValue'])

[ ]: #Transposing filtered_df to have Sample IDs as the rows
genotype_transposed = filtered_df.set_index('ID').transpose()

#Merging the transposed genotype dataframe with the phenotype dataframe on SampleID
merged_df = genotype_transposed.merge(phenotype_df, left_index=True, right_on='SampleID')

# Setting index to 'SampleID'
merged_df.set_index('SampleID', inplace=True)

[ ]: #There were some columns from the parse_vcf function that should not have been included
#(they did not represent SNPs), so we drop them
rs_columns = merged_df.columns[merged_df.columns.str.startswith('rs')]

# Drop columns that do not start with 'rs'
merged_df = merged_df[rs_columns]

[ ]: # Function to drop duplicate columns, keeping the first
def drop_duplicate_columns(df):
    return df.loc[:, ~df.columns.duplicated()]

#Dropping duplicate columns
merged_df = drop_duplicate_columns(merged_df)

[ ]: """
Note: this function can take quite a while to run,
and can be skipped for now if following dataframes are imported
"""

#Batching the dataframe into 4 sections to ensure RAM is not exceeded

#Making sure all columns are strings
merged_df = merged_df.astype(str)

# Calculating the quarter point
```

```

quarter_point = len(merged_df.columns) // 4

# List to store processed columns
processed_columns_first_quarter = []

# Processing the first quarter of the columns
for column in tqdm(merged_df.columns[:quarter_point], desc="Processing First_
↳Quarter"):
    # Split, convert to integers, and sum
    processed_column = merged_df[column].str.split('|', expand=True).
↳astype(int).sum(axis=1)
    processed_columns_first_quarter.append(processed_column)

#Save as df
numeric_df_first_quarter = pd.concat(processed_columns_first_quarter, axis=1)
numeric_df_first_quarter.columns = merged_df.columns[:quarter_point]

#Create a copy to defragment
numeric_df_first_quarter = numeric_df_first_quarter.copy()
numeric_df_first_quarter.to_csv("numeric_df_first_quarter.csv")

```

```

[ ]: """
Note: this function can take quite a while to run,
and can be skipped for now if following dataframes are imported
"""

# List to store processed columns
processed_columns_second_quarter = []

# Calculating the midpoint, avoiding 'PhenotypeValue' if it's the last column
midpoint = len(merged_df.columns) // 2
if 'PhenotypeValue' in merged_df.columns[-1]:
    midpoint = (len(merged_df.columns) - 1) // 2

# Processing the second quarter of the columns (from quarter_point to midpoint)
for column in tqdm(merged_df.columns[quarter_point:midpoint], desc="Processing_
↳Second Quarter"):
    # Split, convert to integers, and sum
    processed_column = merged_df[column].str.split('|', expand=True).
↳astype(int).sum(axis=1)
    processed_columns_second_quarter.append(processed_column)

# Concatenate into a new dataframe for the second quarter
numeric_df_second_quarter = pd.concat(processed_columns_second_quarter, axis=1)
numeric_df_second_quarter.columns = merged_df.columns[quarter_point:midpoint]

numeric_df_second_quarter = numeric_df_second_quarter.copy()

```

```
numeric_df_second_quarter.to_csv("numeric_df_second_quarter.csv")
```

```
[ ]: """
Note: this function can take quite a while to run,
and can be skipped for now if following dataframes are imported
"""

# List to store processed columns for the 3rd quarter
processed_columns_third_quarter = []

# Processing the 3rd quarter of the columns
for column in tqdm(merged_df.columns[quarter_point*2:quarter_point*3],
    desc="Processing Third Quarter"):
    # Split, convert to integers, and sum
    processed_column = merged_df[column].str.split('|', expand=True).
    astype(int).sum(axis=1)
    processed_columns_third_quarter.append(processed_column)

# Save as df
numeric_df_third_quarter = pd.concat(processed_columns_third_quarter, axis=1)
numeric_df_third_quarter.columns = merged_df.columns[quarter_point*2:
    quarter_point*3]

# Create a copy to defragment
numeric_df_third_quarter = numeric_df_third_quarter.copy()
numeric_df_third_quarter.to_csv("numeric_df_third_quarter.csv")
```

```
[ ]: """
Note: this function can take quite a while to run,
and can be skipped for now if following dataframes are imported
"""

# List to store processed columns for the 4th quarter
processed_columns_fourth_quarter = []

# Processing the 4th quarter of the columns
for column in tqdm(merged_df.columns[quarter_point*3:], desc="Processing Fourth
    Quarter"):
    # Split, convert to integers, and sum
    processed_column = merged_df[column].str.split('|', expand=True).
    astype(int).sum(axis=1)
    processed_columns_fourth_quarter.append(processed_column)

# Save as df
numeric_df_fourth_quarter = pd.concat(processed_columns_fourth_quarter, axis=1)
numeric_df_fourth_quarter.columns = merged_df.columns[quarter_point*3:]
```

```
# Create a copy to defragment
numeric_df_fourth_quarter = numeric_df_fourth_quarter.copy()
numeric_df_fourth_quarter.to_csv("numeric_df_fourth_quarter.csv")
```

```
[ ]: """
Note: reading in the csus can take quite a while to run,
and can be skipped for now if the final dataframes are imported
"""

df1 = pd.read_csv("numeric_df_first_quarter.csv")
df2 = pd.read_csv("numeric_df_second_quarter.csv")
df3 = pd.read_csv("numeric_second_half_part1_df.csv")
df4 = pd.read_csv("numeric_second_half_part2_df.csv")
phenotype_df = pd.read_csv('ps3_gwas.phen', sep='\t', header=None,
    names=['SampleID', 'PhenotypeValue'])
```

```
[ ]: #Merging the dataframes
concatenated_df = pd.concat([df1, df2], axis=1)
concatenated_df_final = pd.concat([concatenated_df, df3, df4], axis=1)

#Saving concatenated_df_final as df
#concatenated_df_final.to_csv("concatenated_df_final.csv")
```

```
[4]: #Reading in concatenated_df_final to avoid above functions if possible
newdf = pd.read_csv("concatenated_df_final.csv")
```

```
/tmp/ipykernel_157/1604122887.py:1: DtypeWarning: Columns (840663) have mixed
types. Specify dtype option on import or set low_memory=False.
newdf = pd.read_csv("concatenated_df_final.csv")
```

```
[6]: # Drop the last column of newdf, as the phenotypes can be displayed incorrectly
last_column = newdf.columns[-1]
newdf = newdf.drop(last_column, axis=1)
newdf = drop_duplicate_columns(newdf)

#Merging the SNP data with the phenotypes
newdf = newdf.merge(phenotype_df, on='SampleID', how='left')
```

```
[10]: #Saving new_df to csv
newdf.to_csv("concat_w_phen.csv")
```

## 2 Linear Regression

```
[14]: # Identifying and dropping columns with no variation (excluding 'SampleID' and
↳ 'PhenotypeValue')
snp_columns = [col for col in newdf.columns if col not in ['SampleID',
↳ 'PhenotypeValue']]
columns_to_drop = [col for col in snp_columns if newdf[col].nunique() <= 1]
newdf_cleaned = newdf.drop(columns=columns_to_drop)
```

```
[14]: (8567,
      Unnamed: 0 SampleID  rs11252127  rs7909677  rs11591988  rs12768206 \
0          0  NA06984          0          0          0          1
1          1  NA06989          1          1          0          1
2          2  NA12878          0          0          0          1
3          3  NA18489          0          0          0          1
4          4  NA18504          0          0          0          0

      rs10904561  rs7917054  rs7906287  rs9419557  ...  rs2739260  rs2229949 \
0          0          1          1          0  ...          2          2
1          1          1          2          0  ...          1          2
2          0          1          1          0  ...          1          2
3          0          1          1          0  ...          0          1
4          0          0          0          0  ...          0          0

      rs3750508  rs3750510  rs9777369  rs11137376  rs17583562  rs11137379 \
0          0          2          0          0          0          0
1          1          2          0          0          1          1
2          1          2          0          0          0          0
3          0          2          1          1          0          0
4          0          2          2          0          0          0

      rs9314655  PhenotypeValue
0          0      -1.893857
1          0      2.467882
2          0     -1.565316
3          1     -0.219490
4          2     -0.260466

[5 rows x 832097 columns])
```

```
[17]: """
Note: this function can take quite a while to run,
and can be skipped for now if final dataframes are imported
"""

#Performing linear regression
results_summary = []
```



```

for snp in tqdm(newdf_cleaned.columns[3:-1]): # Adjust the slice as necessary
    ↪to skip non-SNP columns
    # Ensure there's variation in SNP data
    if newdf_cleaned[snp].nunique() > 1:
        X = sm.add_constant(newdf_cleaned[snp]) # SNP data as independent
    ↪variable
        y = newdf_cleaned['PhenotypeValue'] # Phenotype as dependent variable

        # Fit the model
        model = sm.OLS(y, X, missing='drop') # 'missing='drop'' to handle
    ↪missing values
        result = model.fit()

        if result.pvalues.shape[0] > 1: # Check if SNP coefficient exists
            summary = {
                'SNP': snp,
                'p-value': result.pvalues[1], # p-value for SNP coefficient
                'beta': result.params[1] # Beta coefficient for SNP
            }
            results_summary.append(summary)
        else:
            print(f"Model fitting issue with SNP {snp}. Likely due to constant
    ↪SNP values after dropping missing data.")
        else:
            print(f"No variation in SNP {snp}. Skipping.")

results_df = pd.DataFrame(results_summary)

```

100%| | 832093/832093 [27:28<00:00, 504.66it/s]

[18]: results\_df

```

[18]:
      SNP  p-value  beta
0  rs7909677  0.363471  0.191564
1  rs11591988  0.504264 -0.146990
2  rs12768206  0.547052  0.057168
3  rs10904561  0.142948 -0.178488
4  rs7917054  0.547052  0.057168
...
832088  rs9777369  0.000822  0.505054
832089  rs11137376  0.034320  0.361115
832090  rs17583562  0.001052 -0.712181
832091  rs11137379  0.319168 -0.139623
832092  rs9314655  0.000044  0.314990

```

[832093 rows x 3 columns]

```
[19]: results_df_sorted = results_df.sort_values(by='p-value', ascending=False)

results_df_sorted
```

```
[19]:
```

	SNP	p-value	beta
728239	rs1319484	9.999937e-01	9.697146e-07
660203	rs3131012	9.999903e-01	1.178195e-06
660201	rs2240063	9.999903e-01	1.178195e-06
660200	rs2240064	9.999903e-01	1.178195e-06
797823	rs7041298	9.999902e-01	1.605079e-06
...	...	...	...
830288	rs507666	5.675715e-13	-9.908198e-01
830286	rs2519093	5.675715e-13	-9.908198e-01
300402	rs1531517	2.327750e-15	1.000888e+00
300406	rs4803750	6.102444e-29	1.898317e+00
300403	rs62117204	2.121923e-32	2.104528e+00

[832093 rows x 3 columns]

```
[16]: #Saving final linear regression results to csv
#results_df_sorted.to_csv("linRegResults.csv")
#Reading in results_df_sorted
results_df_sorted = pd.read_csv("linRegResults.csv")
```

### 3 Metrics

```
[13]: #Reviewing metrics of our linear regression compared to the output of Plink
plinkres = pd.read_csv("ps3_gwas.assoc.linear", delim_whitespace=True)
```

```
[14]: plinkres.head()
```

```
[14]:
```

	CHR	SNP	BP	A1	TEST	NMISS	BETA	STAT	P
0	10	rs11252127	98087	T	ADD	207	-0.20430	-1.6060	0.1097
1	10	rs7909677	111955	G	ADD	207	0.19160	0.9108	0.3635
2	10	rs11591988	126070	T	ADD	207	-0.14700	-0.6690	0.5043
3	10	rs12768206	134767	A	ADD	207	-0.05717	-0.6032	0.5471
4	10	rs10904561	135656	G	ADD	207	-0.17850	-1.4710	0.1429

```
[18]: #Merging our linear regression results with the plink results on "SNP"
linregcompare = pd.merge(results_df_sorted[['SNP', 'p-value', 'beta']],
    ↪ plinkres[['SNP', 'P', 'BETA']], on='SNP', suffixes=('_pred', '_true'))

#Calculating differences between pred and true values
linregcompare['p_value_diff'] = abs(linregcompare['p-value'] -
    ↪ linregcompare['P'])
linregcompare['beta_diff'] = abs(linregcompare['beta'] - linregcompare['BETA'])
```

```
[20]: # Computing MAE, MSE, and RMSE

# For p-values
mae_p_value = np.mean(linregcompare['p_value_diff'])
mse_p_value = np.mean(linregcompare['p_value_diff']**2)
rmse_p_value = np.sqrt(mse_p_value)

# For beta coefficients
mae_beta = np.mean(linregcompare['beta_diff'])
mse_beta = np.mean(linregcompare['beta_diff']**2)
rmse_beta = np.sqrt(mse_beta)

# Print the results
print(f"MAE for p-values: {mae_p_value}")
print(f"MSE for p-values: {mse_p_value}")
print(f"RMSE for p-values: {rmse_p_value}")

print(f"MAE for beta coefficients: {mae_beta}")
print(f"MSE for beta coefficients: {mse_beta}")
print(f"RMSE for beta coefficients: {rmse_beta}")
```

```
MAE for p-values: 0.0001235241545070243
MSE for p-values: 5.163739526631637e-05
RMSE for p-values: 0.0071859164527787525
MAE for beta coefficients: 0.11027235180768107
MSE for beta coefficients: 0.059025047364823356
RMSE for beta coefficients: 0.24295070974340321
```

```
[ ]:
```