**##chunking file**

pip install pandas

import pandas as pd

vcf\_file = "/media/puneet/SVM/1001genomes\_snp-short-indel\_only\_ACGTN.vcf"

print(vcf\_file)

import os

cmd = "zgrep '^#' " +vcf\_file + "|tail -n 1"

columns = os.popen(cmd).read()

columns

columns = columns.strip('#')

columns

columns = columns.split('\t')

print(columns)

len(columns)

pip install pysam

import pysam

chunk\_size = 100000

output\_prefix = "vcf\_chunk\_"

with pysam.VariantFile(vcf\_file) as vcf\_in:

header = vcf\_in.header

chunk = []

for i, record in enumerate(vcf\_in):

chunk.append(record)

if (i + 1) % chunk\_size == 0:

chunk\_file = f"{output\_prefix}{i // chunk\_size + 1}.vcf"

with pysam.VariantFile(chunk\_file, 'w', header=header) as vcf\_out:

for rec in chunk:

vcf\_out.write(rec)

chunk = []

**# Write remaining records**

if chunk:

chunk\_file = f"{output\_prefix}{i // chunk\_size + 1}.vcf"

with pysam.VariantFile(chunk\_file, 'w', header=header) as vcf\_out:

for rec in chunk:

vcf\_out.write(rec)

import pysam

import pandas as pd

chunk\_file = "vcf\_chunk\_88.vcf"

**# Read the VCF chunk**

with pysam.VariantFile(chunk\_file) as vcf\_in:

records = []

for record in vcf\_in:

rec\_dict = {

'chrom': record.chrom,

'pos': record.pos,

'id': record.id,

'ref': record.ref,

'alt': ','.join(str(a) for a in record.alts),

'qual': record.qual,

'filter': ','.join(record.filter.keys()),

'info': record.info

}

records.append(rec\_dict)

**# Convert to DataFrame**

df = pd.DataFrame(records)

# Example analysis: summary statistics

summary = df.describe()

print(summary)

df.head()

df.tail()

##execution of ML model

import pandas as pd

file\_path = "/media/puneet/SVM/annoteted files/SMCS\_PSI\_1001.csv"

import io

df = pd.read\_csv(file\_path, sep=',')

df.head()

print(df)

df.columns

df.describe()

import matplotlib.pyplot as plt

import seaborn as sns

column\_name = 'variants\_effect\_downstream\_gene\_variant'

**# Plot the distribution**

plt.figure(figsize=(10, 6))

sns.histplot(df[column\_name], kde=True, bins=30)

plt.title(f'Distribution of {column\_name}')

plt.xlabel(column\_name)

plt.ylabel('Frequency')

plt.show()

col\_name = 'variants\_effect\_intron\_variant'

# Plot the distribution

plt.figure(figsize=(10, 6))

sns.histplot(df[col\_name], kde=True, bins=30)

plt.title(f'Distribution of {col\_name}')

plt.xlabel(column\_name)

plt.ylabel('Frequency')

plt.show()

# Select only numeric columns

numeric\_df = df.select\_dtypes(include='number')

correlation\_matrix = numeric\_df.corr()

plt.figure(figsize=(12, 10))

sns.heatmap(correlation\_matrix, annot=True, fmt='.2f', cmap='coolwarm', linewidths=.5)

plt.show()

#TRIAL,

AFTER CLOSELY WATCHING THE DATA I GET TO KNOW THAT SOME GENENAMES ARE LIKE GENEID ONLY

reference\_df = pd.read\_csv("/media/puneet/SVM/puneet-smcs/reference.csv")

reference\_df.head()

target\_df = pd.read\_csv('/home/puneet/Downloads/SMCS\_PSI\_1001.csv')

target\_df.head()

gene\_id\_to\_name = dict(zip(reference\_df['GeneId'], reference\_df['GeneName']))

def replace\_gene\_name(row):

if row['GeneId'] in gene\_id\_to\_name:

return gene\_id\_to\_name[row['GeneId']]

else:

return row['GeneName']

target\_df['GeneName'] = target\_df.apply(replace\_gene\_name, axis=1)

# Save the updated target file

target\_df.to\_csv('updated\_target1.csv', index=False)

print("Gene names have been updated successfully.")

#NOW HANDLING MISSING VALUES IN BIOTYPE

df3 = pd.read\_csv('/media/puneet/SVM/puneet-smcs/SMCS\_1001\_upd.csv')

# Replace empty strings with NaN

df3['BioType'].replace('', pd.NA, inplace=True)

# Fill missing values with 'protein\_coding'

df3['BioType'].fillna('protein\_coding', inplace=True)

**# Save the updated DataFrame to a new CSV file**

df3.to\_csv('genome\_dataset\_filled.csv', index=False)

cleaned\_file = pd.read\_csv("/media/puneet/SVM/puneet-smcs/final\_cleaned - genome\_dataset\_filled.csv")

cleaned\_file.describe

cleaned\_file.head()

impact\_summary = cleaned\_file[['variants\_impact\_HIGH', 'variants\_impact\_MODERATE', 'variants\_impact\_LOW', 'variants\_impact\_MODIFIER']].sum()

print("Impact Distribution Summary:")

print(impact\_summary)

**# Visualize the impact distribution**

plt.figure(figsize=(10, 6))

sns.barplot(x=impact\_summary.index, y=impact\_summary.values)

plt.title('Distribution of Variants by Impact')

plt.xlabel('Impact Type')

plt.ylabel('Number of Variants')

plt.show()

high\_impact\_genes = cleaned\_file.groupby('GeneName')['variants\_impact\_HIGH'].sum().sort\_values(ascending=False)

top\_high\_impact\_genes = high\_impact\_genes.head(10) # Adjust the number to get more/fewer genes

print("\nTop Genes with Highest Number of High-Impact Variants:")

print(top\_high\_impact\_genes)

# Visualize the top genes with the highest number of high-impact variants

plt.figure(figsize=(12, 8))

sns.barplot(x=top\_high\_impact\_genes.values, y=top\_high\_impact\_genes.index, palette='viridis')

plt.title('Top Genes with Highest Number of High-Impact Variants')

plt.xlabel('Number of High-Impact Variants')

plt.ylabel('Gene Name')

plt.show()

Low\_impact\_genes = cleaned\_file.groupby('GeneName')['variants\_impact\_LOW'].sum().sort\_values(ascending=False)

Top\_Low\_impact\_genes = Low\_impact\_genes.head(10)

**# Visualize the top genes with the highest number of high-impact variants**

plt.figure(figsize=(12, 8))

sns.barplot(x=Top\_Low\_impact\_genes, y=Top\_Low\_impact\_genes.index, palette='inferno')

plt.title('Top Genes with Highest Number of LOW-Impact Variants')

plt.xlabel('Number of LOW-Impact Variants')

plt.ylabel('Gene Name')

plt.show()

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

import seaborn as sns

data = pd.read\_csv('/media/puneet/SVM/puneet-smcs/final\_cleaned - genome\_dataset\_filled.csv')

clustering\_data = data[['variants\_impact\_HIGH', 'variants\_impact\_LOW', 'variants\_impact\_MODERATE', 'variants\_impact\_MODIFIER']]

scaler = StandardScaler()

clustering\_data\_scaled = scaler.fit\_transform(clustering\_data)

n\_clusters = 4

kmeans = KMeans(n\_clusters=n\_clusters, random\_state=42)

clusters = kmeans.fit\_predict(clustering\_data\_scaled)

data['Cluster'] = clusters

pca = PCA(n\_components=2)

pca\_components = pca.fit\_transform(clustering\_data\_scaled)

data['PCA1'] = pca\_components[:, 0]

data['PCA2'] = pca\_components[:, 1]

cluster\_summary = data.groupby('Cluster')[['variants\_impact\_HIGH', 'variants\_impact\_LOW', 'variants\_impact\_MODERATE', 'variants\_impact\_MODIFIER']].mean()

print(cluster\_summary)

cluster\_names = {

0: 'High Impact',

1: 'Low Impact',

2: 'Moderate Impact',

3: 'Modifier Impact'

}

data['ClusterName'] = data[‘Cluster'].map(cluster\_names)

**# Visualize the clusters**

plt.figure(figsize=(10, 6))

sns.scatterplot(x='PCA1', y='PCA2', hue='ClusterName', data=data, palette='viridis', s=100, alpha=0.6)

plt.title('Clusters of Genes based on Impact Severity')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.legend(title='Cluster')

plt.show()

# Select only numeric columns

numeric\_df = data .select\_dtypes(include='number')

correlation\_matrix = numeric\_df.corr()

plt.figure(figsize=(12, 10))

sns.heatmap(correlation\_matrix, annot=True, fmt='.2f', cmap='coolwarm', linewidths=.5)

plt.show()

import pandas as pd

**# Print column names**

print(df.columns)

**# Strip whitespace from column names**

df.columns = df.columns.str.strip()

**# Check for the exact column name**

if 'GeneName' in df.columns:

print("Column GeneName exists")

else:

print("Column GeneName does not exist")

import pandas as pd

**# Display the first few rows of the dataset to understand its structure**

print(df.head())

print(df.info())

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, LabelEncoder

**# Drop any irrelevant or empty columns**

data = df.drop(columns=['Unnamed: 34'])

**# Encode categorical variables**

label\_encoders = {}

for column in ['GeneName', 'GeneId', 'TranscriptId', 'BioType']:

label\_encoders[column] = LabelEncoder()

data[column] = label\_encoders[column].fit\_transform(data[column])

**# Define feature columns and target columns**

feature\_columns = data.columns.difference(['GeneName', 'GeneId', 'BioType', 'TranscriptId', 'variants\_impact\_HIGH'])

target\_column = 'variants\_impact\_HIGH'

**# Split the data into training and testing sets**

X = data[feature\_columns]

y = data[target\_column]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**# Standardize the features**

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

**# Check the shapes of the resulting datasets**

print(f"X\_train shape: {X\_train.shape}")

print(f"X\_test shape: {X\_test.shape}")

print(f"y\_train shape: {y\_train.shape}")

print(f"y\_test shape: {y\_test.shape}")

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, accuracy\_score

**# Initialize the model**

model = RandomForestClassifier(n\_estimators=100, random\_state=42, class\_weight='balanced')

**# Train the model**

model.fit(X\_train, y\_train)

**# Make predictions**

y\_pred = model.predict(X\_test)

**# Evaluate the model**

accuracy = accuracy\_score(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred)

print(f"Accuracy: {accuracy}")

print(f"Classification Report:\n{report}")

import matplotlib.pyplot as plt

import numpy as np

from sklearn.model\_selection import learning\_curve

train\_sizes, train\_scores, test\_scores = learning\_curve(

estimator=model, X=X\_train, y=y\_train, train\_sizes=np.linspace(0.1, 1.0, 10),

cv=5, scoring='accuracy', n\_jobs=-1)

train\_scores\_mean = np.mean(train\_scores, axis=1)

train\_scores\_std = np.std(train\_scores, axis=1)

test\_scores\_mean = np.mean(test\_scores, axis=1)

test\_scores\_std = np.std(test\_scores, axis=1)

plt.figure(figsize=(10, 6))

plt.fill\_between(train\_sizes, train\_scores\_mean - train\_scores\_std,

train\_scores\_mean + train\_scores\_std, alpha=0.1,

color="r")

plt.fill\_between(train\_sizes, test\_scores\_mean - test\_scores\_std,

test\_scores\_mean + test\_scores\_std, alpha=0.1, color="g")

plt.plot(train\_sizes, train\_scores\_mean, 'o-', color="r",

label="Training score")

plt.plot(train\_sizes, test\_scores\_mean, 'o-', color="g",

label="Cross-validation score")

plt.xlabel("Training examples")

plt.ylabel("Accuracy Score")

plt.legend(loc="best")

plt.title("Learning Curve (Random Forest)")

plt.show()