In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
# Define column names (First 6 columns are metadata, rest are genotypic data)
col_names = ["Family_ID", "Individual_ID", "Paternal_ID", "Maternal_ID", "Sex", "Phenotype"]

# Loads only first 6 metadata columns
df_metadata = pd.read_csv("donors.ped", delim_whitespace=True, names=col_names, usecols=range(6))

# Display first few rows
print(df_metadata.head())
```

```
   Family_ID  Individual_ID  Paternal_ID  Maternal_ID  Sex  Phenotype
0          1           4023            0            0    0         -9
1          1           4313            0            0    0         -9
2          1           4054            0            0    0         -9
3          1           4165            0            0    0         -9
4          1           4373            0            0    0         -9
```

In [3]:
```
df_full = pd.read_csv("donors.ped", delim_whitespace=True, header=None)

# Assign column names (first 6 are metadata, rest are SNPs)
metadata_cols = ["Family_ID", "Individual_ID", "Paternal_ID", "Maternal_ID", "Sex", "Phenotype"]
snp_cols = [f"SNP_{i}" for i in range(1, len(df_full.columns) - 5)]  # SNPs start from column 7
df_full.columns = metadata_cols + snp_cols

# Display first few rows
print(df_full.head())
```

```
   Family_ID  Individual_ID  Paternal_ID  Maternal_ID  Sex  Phenotype SNP_1  \
0          1           4023            0            0    0         -9     G
1          1           4313            0            0    0         -9     A
2          1           4054            0            0    0         -9     G
3          1           4165            0            0    0         -9     G
4          1           4373            0            0    0         -9     G

  SNP_2 SNP_3 SNP_4  ... SNP_26507 SNP_26508 SNP_26509 SNP_26510 SNP_26511  \
0     A     G     A  ...         A         A         G         A         A
1     A     G     A  ...         A         A         A         A         A
2     A     A     A  ...         A         A         A         A         A
3     A     G     A  ...         A         A         G         G         A
4     A     G     A  ...         A         A         A         A         A

  SNP_26512 SNP_26513 SNP_26514 SNP_26515 SNP_26516
0         A         G         G         G         G
1         A         G         G         G         G
2         A         G         G         G         G
3         A         G         G         G         G
4         A         G         G         G         G

[5 rows x 26522 columns]
```

In [4]:
```python
print(df_full.info())  # Check column types
print(df_full.describe())  # Get summary statistics
print(df_full.isnull().sum())  # Check for missing values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 986 entries, 0 to 985
Columns: 26522 entries, Family_ID to SNP_26516
dtypes: int64(6), object(26516)
memory usage: 199.5+ MB
None
        Family_ID  Individual_ID  Paternal_ID  Maternal_ID    Sex  Phenotype
count  986.000000     986.000000        986.0        986.0  986.0      986.0
mean    47.805274    4497.562880          0.0          0.0    0.0       -9.0
std     27.105333     288.401283          0.0          0.0    0.0        0.0
min      1.000000    4000.000000          0.0          0.0    0.0       -9.0
25%     24.000000    4247.250000          0.0          0.0    0.0       -9.0
50%     48.000000    4501.500000          0.0          0.0    0.0       -9.0
75%     70.000000    4747.750000          0.0          0.0    0.0       -9.0
max     96.000000    4994.000000          0.0          0.0    0.0       -9.0
Family_ID        0
Individual_ID    0
Paternal_ID      0
Maternal_ID      0
Sex              0
                ..
SNP_26512        0
SNP_26513        0
SNP_26514        0
SNP_26515        0
SNP_26516        0
Length: 26522, dtype: int64
```

In [8]:
```
# Select only SNP columns
snp_data = df_full.iloc[:, 6:]  # Excluding metadata columns

# Count unique values (alleles) for each SNP
allele_counts = snp_data.apply(lambda col: col.value_counts())

# Display allele frequencies for first 5 SNPs
print(allele_counts.head())
```

```
   SNP_1  SNP_2  SNP_3  SNP_4  SNP_5  SNP_6  SNP_7  SNP_8  SNP_9  SNP_10  ...  \
0    2.0    2.0    NaN    NaN    1.0    1.0    8.0    8.0    1.0     1.0  ...
A  345.0  816.0  340.0  832.0  428.0  858.0  551.0  120.0   76.0     2.0  ...
C    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN     NaN  ...
G  639.0  168.0  646.0  154.0  557.0  127.0  427.0  858.0  909.0   983.0  ...
T    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN     NaN  ...

   SNP_26507  SNP_26508  SNP_26509  SNP_26510  SNP_26511  SNP_26512  \
0        NaN        NaN        1.0        1.0        6.0        6.0
A      757.0      967.0      253.0      724.0      895.0      977.0
C        NaN        NaN        NaN        NaN        NaN        NaN
G      229.0       19.0      732.0      261.0       85.0        3.0
T        NaN        NaN        NaN        NaN        NaN        NaN

   SNP_26513  SNP_26514  SNP_26515  SNP_26516
0        NaN        NaN        3.0        3.0
A      228.0       20.0       59.0        1.0
C        NaN        NaN        NaN        NaN
G      758.0      966.0      924.0      982.0
T        NaN        NaN        NaN        NaN

[5 rows x 26516 columns]
```

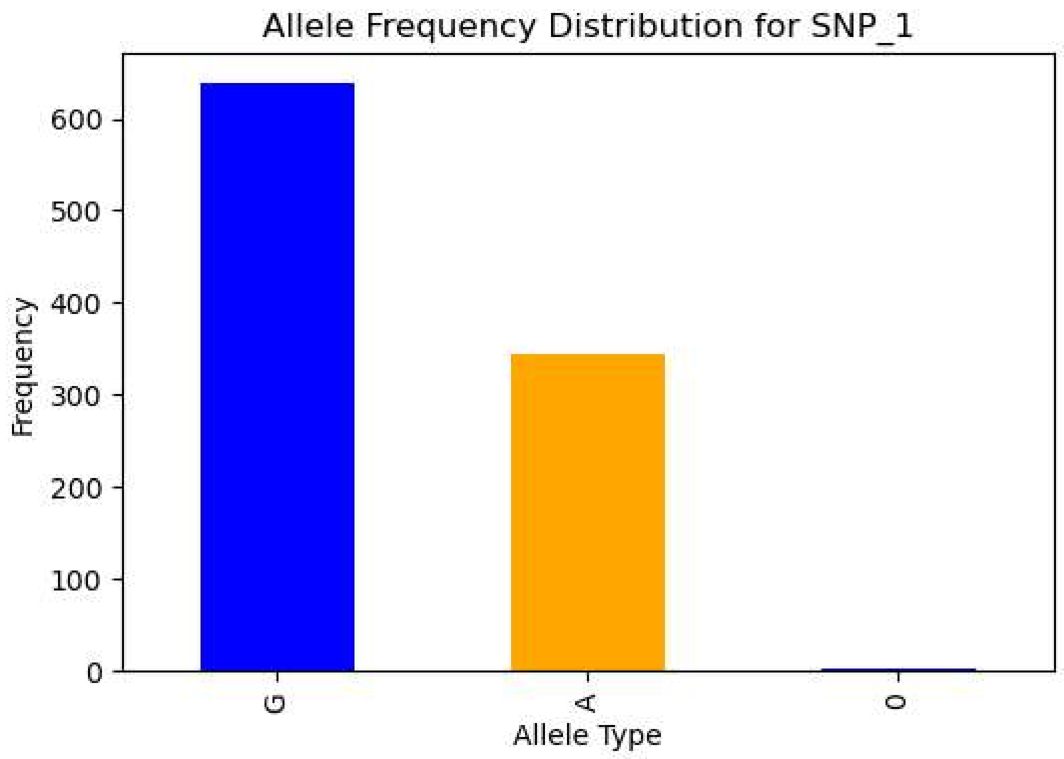
In [9]:

```python
# Count missing values in the dataset
missing_values = df_full.isnull().sum()

# Print columns with missing values
print(missing_values[missing_values > 0])
```

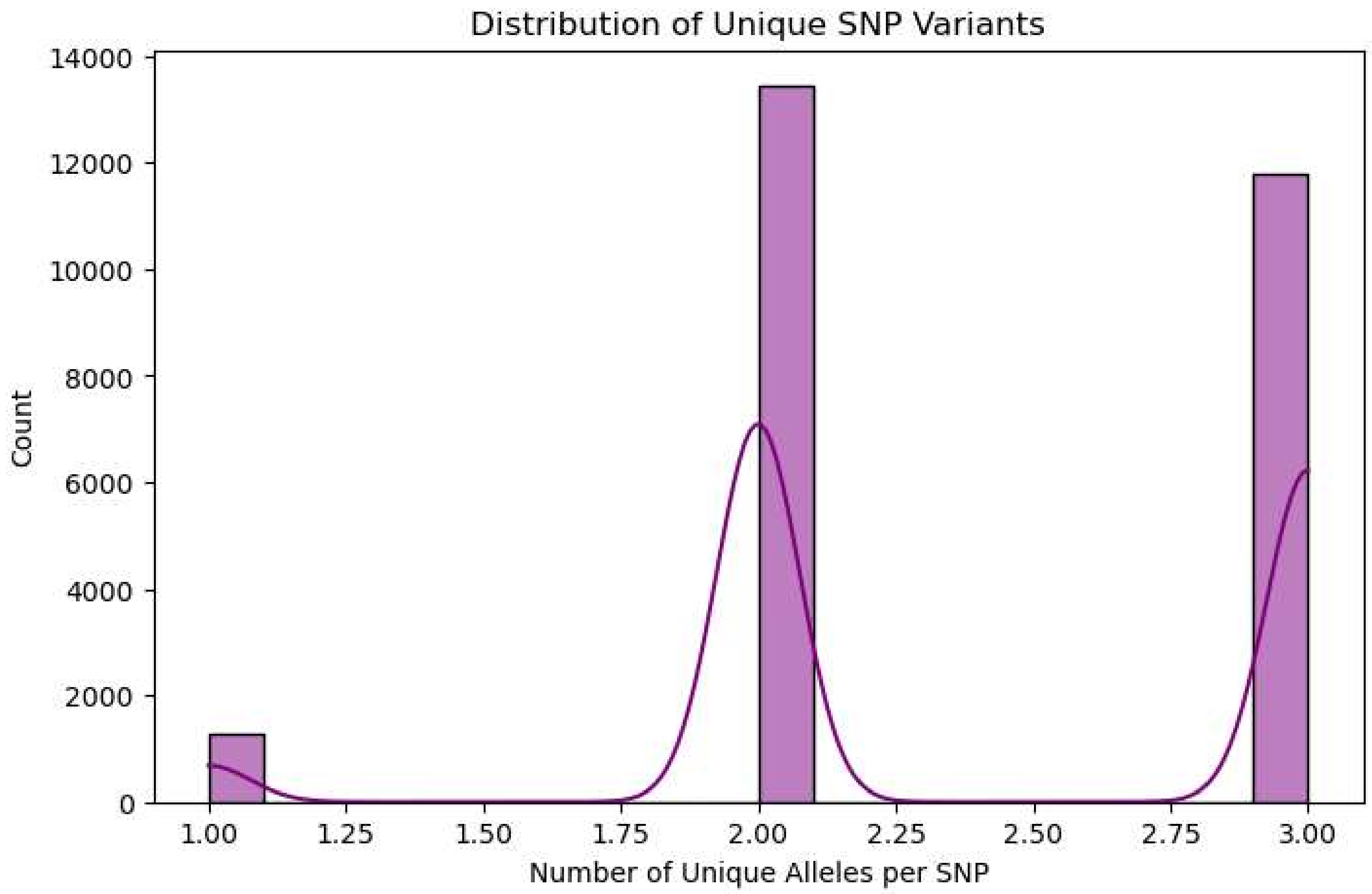
```
Series([], dtype: int64)
```

In [10]:

```python
import matplotlib.pyplot as plt

# Get the first SNP column (example)
snp_column = snp_data.columns[0]

# Count allele occurrences
allele_counts = snp_data[snp_column].value_counts()

# Plot
plt.figure(figsize=(6, 4))
allele_counts.plot(kind='bar', color=['blue', 'orange'])
plt.xlabel("Allele Type")
plt.ylabel("Frequency")
plt.title(f"Allele Frequency Distribution for {snp_column}")
plt.show()
```



In [11]:
```
import seaborn as sns

# Count the number of unique alleles per SNP
num_unique_alleles = snp_data.nunique()

# Plot distribution
plt.figure(figsize=(8, 5))
sns.histplot(num_unique_alleles, bins=20, kde=True, color="purple")
plt.xlabel("Number of Unique Alleles per SNP")
plt.ylabel("Count")
plt.title("Distribution of Unique SNP Variants")
plt.show()
```



In [12]:
```
df_full.to_csv("processed_genetic_data.csv", index=False)
print("Processed genetic data saved as 'processed_genetic_data.csv'.")
```

```
Processed genetic data saved as 'processed_genetic_data.csv'.
```

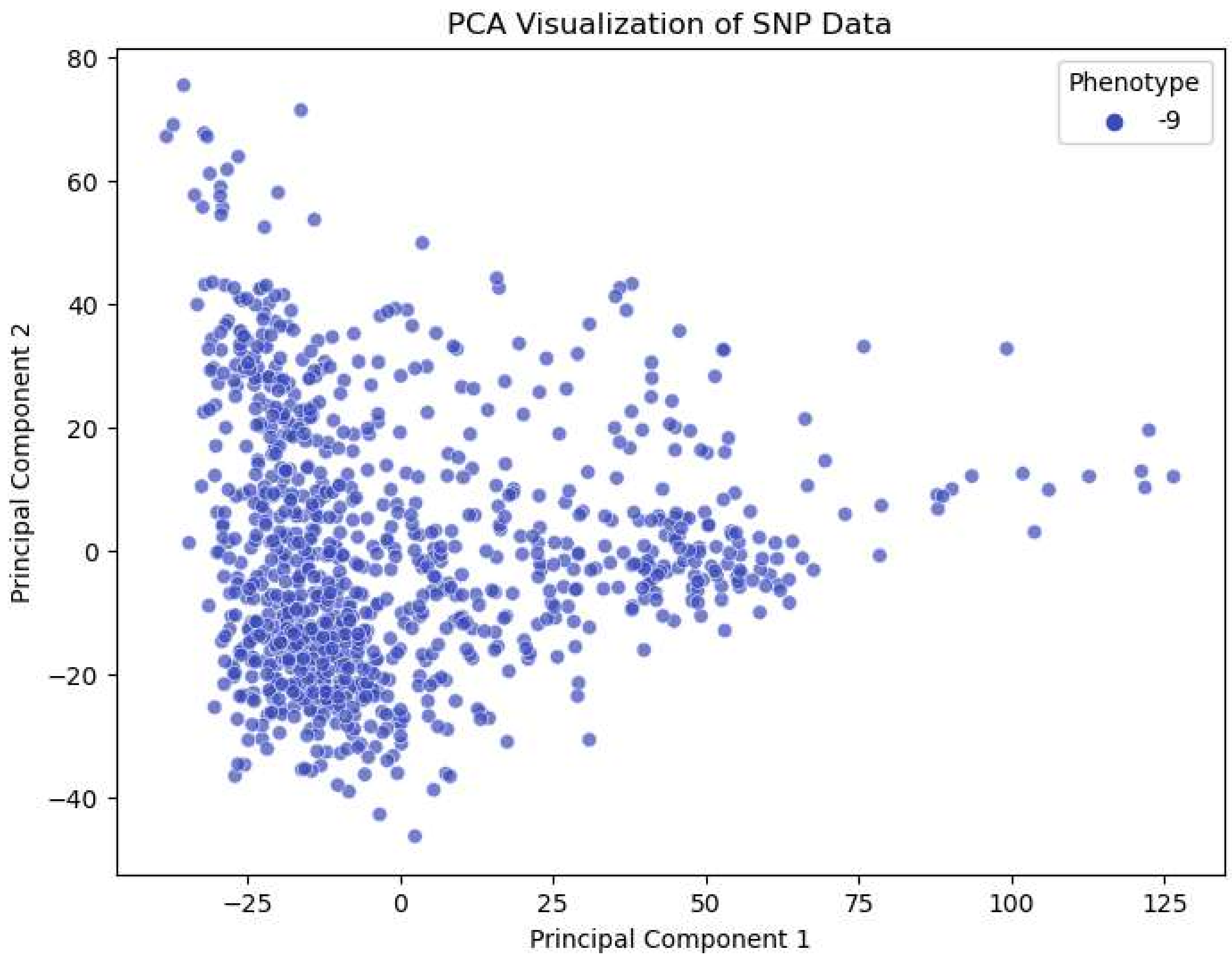
In [13]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder, StandardScaler
import numpy as np

# Select only SNP columns (excluding metadata)
snp_data = df_full.iloc[:, 6:].copy()

# Convert allele pairs into numeric values (A/G → 0, G/G → 1, A/A → 2)
def encode_snp(col):
    unique_vals = col.unique()
    mapping = {val: i for i, val in enumerate(unique_vals)}
    return col.map(mapping)

# Apply encoding to all SNP columns
snp_encoded = snp_data.apply(encode_snp)

# Standardize data before PCA
scaler = StandardScaler()
snp_scaled = scaler.fit_transform(snp_encoded)

print("SNP data successfully encoded and standardized!")
```

```
SNP data successfully encoded and standardized!
```

In [14]:
```
# Apply PCA
pca = PCA(n_components=2)
pca_result = pca.fit_transform(snp_scaled)

# Convert to DataFrame
df_pca = pd.DataFrame(pca_result, columns=["PC1", "PC2"])

# Add metadata for visualization
df_pca["Phenotype"] = df_full["Phenotype"]

print(df_pca.head())  # Show first few PCA-transformed rows
```

```
         PC1        PC2  Phenotype
0  37.905687  -2.344110         -9
1 -17.132511  20.429012         -9
2  40.269471  -2.201245         -9
3  55.590880  -5.874212         -9
4 -21.472823  40.152571         -9
```

In [15]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 6))
sns.scatterplot(x="PC1", y="PC2", hue=df_pca["Phenotype"], palette="coolwarm", alpha=0.7, data=df_pca)

plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA Visualization of SNP Data")
plt.legend(title="Phenotype")
plt.show()
```



In [16]:
```
explained_variance = pca.explained_variance_ratio_
print(f"PC1 explains {explained_variance[0]*100:.2f}% of the variance")
print(f"PC2 explains {explained_variance[1]*100:.2f}% of the variance")
```

```
PC1 explains 3.07% of the variance
PC2 explains 1.69% of the variance
```

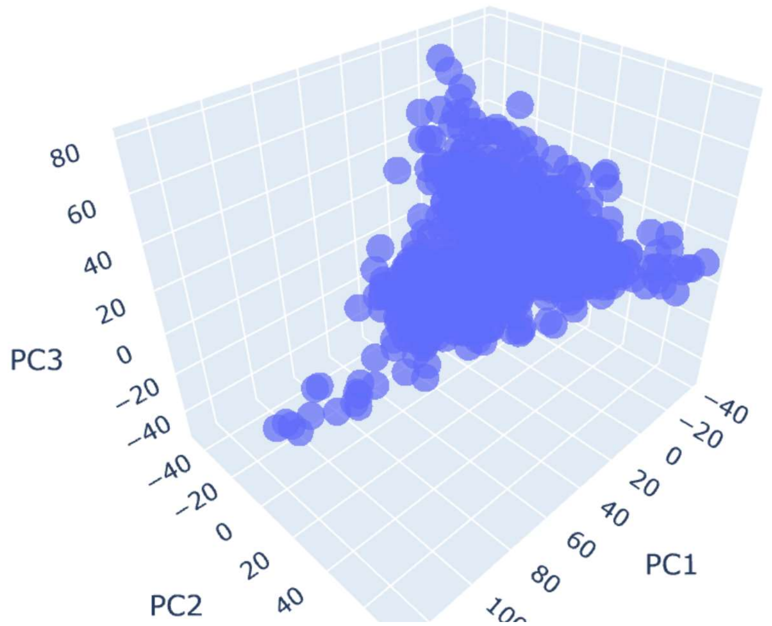
In [17]:
```
from sklearn.decomposition import PCA

# Apply PCA with 3 components
pca_3d = PCA(n_components=3)
```

```
pca_result_3d = pca_3d.fit_transform(snp_scaled)

# Convert to DataFrame
df_pca_3d = pd.DataFrame(pca_result_3d, columns=["PC1", "PC2", "PC3"])

# Add metadata for visualization
df_pca_3d["Phenotype"] = df_full["Phenotype"]

print(df_pca_3d.head())  # Show first few PCA-transformed rows
```

```
         PC1        PC2        PC3  Phenotype
0  37.905747  -2.336705 -19.484211         -9
1 -17.132508  20.434945  35.855234         -9
2  40.269437  -2.206716  -2.525267         -9
3  55.590880  -5.867082  -8.748254         -9
4 -21.472791  40.142954  -4.625787         -9
```

In [18]:
```
import plotly.express as px

# Create a 3D scatter plot
fig = px.scatter_3d(df_pca_3d, x="PC1", y="PC2", z="PC3",
                    color=df_pca_3d["Phenotype"].astype(str),  # Convert phenotype to string for color coding
                    title="3D PCA Visualization of SNP Data",
                    labels={"Phenotype": "Phenotype"},
                    opacity=0.7)

fig.show()
```

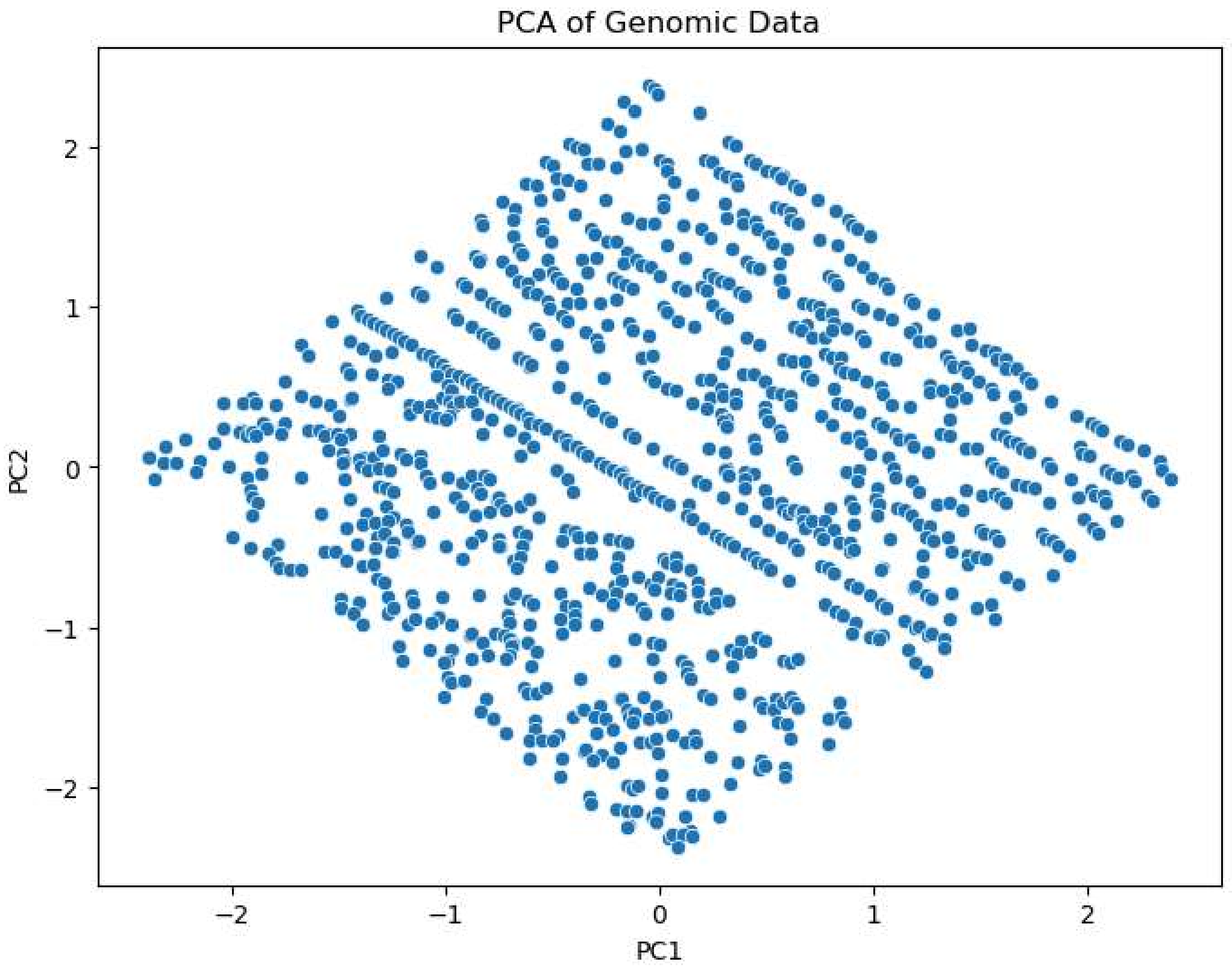
3D PCA Visualization of SNP Data



```
In [19]: explained_variance = pca_3d.explained_variance_ratio_
         print(f"PC1 explains {explained_variance[0]*100:.2f}% of the variance")
         print(f"PC2 explains {explained_variance[1]*100:.2f}% of the variance")
         print(f"PC3 explains {explained_variance[2]*100:.2f}% of the variance")
```

```
PC1 explains 3.07% of the variance
PC2 explains 1.69% of the variance
PC3 explains 1.34% of the variance
```

In [20]:
```
# Standardize the data before PCA
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df_full.select_dtypes(include=[np.number]))

# Apply PCA
pca = PCA(n_components=2)
pca_result = pca.fit_transform(scaled_data)

# Convert to DataFrame for visualization
pca_df = pd.DataFrame(data=pca_result, columns=['PC1', 'PC2'])

# Scatter plot of PCA results
plt.figure(figsize=(8, 6))
sns.scatterplot(x=pca_df['PC1'], y=pca_df['PC2'])
plt.title("PCA of Genomic Data")
plt.show()
```



```
In [ ]: import seaborn as sns
        import matplotlib.pyplot as plt

        # Compute SNP correlation matrix
        snp_corr = snp_encoded.corr()

        # Plot heatmap
        plt.figure(figsize=(12, 8))
        sns.heatmap(snp_corr, cmap="coolwarm", linewidths=0.5, vmin=-1, vmax=1)
```

```
plt.title("SNP Correlation Heatmap")
plt.show()
```

In [ ]:
```
import numpy as np

# Compute pairwise LD (r²) manually
ld_matrix = np.corrcoef(snp_encoded.T) ** 2  # Squared correlation for LD

# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(ld_matrix, cmap="viridis", linewidths=0.5)
plt.title("Linkage Disequilibrium (LD) Matrix")
plt.xlabel("SNPs")
plt.ylabel("SNPs")
plt.show()
```

In [ ]:
```
import numpy as np
from sklearn.decomposition import PCA

# Run PCA on SNP data
pca = PCA(n_components=2)
pca_result = pca.fit_transform(snp_scaled)

# Get PCA loadings (SNP influence)
loadings = pca.components_.T

# Scatter plot for PCA
plt.figure(figsize=(8, 6))
plt.scatter(loadings[:, 0], loadings[:, 1], alpha=0.7, color="red")
plt.xlabel("PC1 Loadings")
plt.ylabel("PC2 Loadings")
plt.title("PCA Biplot: SNP Influence")
plt.show()
```

In [ ]:
```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Test different K values (number of clusters)
inertia = []
K_range = range(1, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(snp_scaled)
    inertia.append(kmeans.inertia_)

# Plot Elbow Curve
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia, marker="o", linestyle="--")
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Inertia")
plt.title("Elbow Method for Optimal K")
plt.show()
```

In [ ]:

```
# Choose the optimal K (e.g., from elbow method)
optimal_k = 3

# Run K-Means
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
snp_clusters = kmeans.fit_predict(snp_scaled)

# Add cluster labels to dataframe
df_full["Cluster"] = snp_clusters

print(df_full[["Phenotype", "Cluster"]].head())  # Check assigned clusters
```

In [ ]:

```
import seaborn as sns

plt.figure(figsize=(8, 6))
sns.scatterplot(x=pca_result[:, 0], y=pca_result[:, 1], hue=df_full["Cluster"], palette="viridis", alpha=0.7)
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("K-Means Clustering of SNPs in PCA Space")
plt.legend(title="Cluster")
plt.show()
```

In [ ]:

```
from sklearn.cluster import DBSCAN

# Run DBSCAN
dbscan = DBSCAN(eps=2, min_samples=5)  # Adjust eps for better separation
db_clusters = dbscan.fit_predict(snp_scaled)

# Add DBSCAN clusters to dataframe
df_full["DBSCAN_Cluster"] = db_clusters

# Visualize in PCA space
plt.figure(figsize=(8, 6))
sns.scatterplot(x=pca_result[:, 0], y=pca_result[:, 1], hue=df_full["DBSCAN_Cluster"], palette="coolwarm", alpha=0.7)
```

```
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("DBSCAN Clustering of SNPs in PCA Space")
plt.legend(title="Cluster")
plt.show()
```

In [ ]: