GWAS simulation explanation

```
# Requirements (install if needed):
# pip install pandas numpy statsmodels matplotlib

import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import os

# ---------- Config ----------
PHENOS_PATH = r"C:\Users\srush\Documents\Desktop\cl2\bi\data\phenos.csv"      # input phenotypes file
OUT_DIR = r"C:\Users\srush\Documents\Desktop\cl2\bi\data\Bioinformatics"
N_SNPS = 5000                          # number of simulated SNPs
TRAITS = ["Yield", "Plant_height", "Grain_weight"]  # traits to analyze
np.random.seed(42)
# ----------------------------

os.makedirs(OUT_DIR, exist_ok=True)

# Load phenotype data
phenos = pd.read_csv(PHENOS_PATH).reset_index(drop=True)

# Check traits exist
for t in TRAITS:
    if t not in phenos.columns:
        raise ValueError(f"Trait '{t}' not found in phenos.csv. Columns: {phenos.columns.tolist()}")

n_samples = phenos.shape[0]
print(f"Samples: {n_samples}, Traits: {TRAITS}, Simulating {N_SNPS} SNPs...")

# Simulate SNP matrix (0,1,2) — rows = samples, cols = SNPs
G = pd.DataFrame(
    np.random.randint(0, 3, size=(n_samples, N_SNPS)),
    columns=[f"SNP_{i+1}" for i in range(N_SNPS)]
)

# Specify causal SNP indices for demonstration (0-based indices)
causal_snps_for_trait = {
    "Yield": [10, 150, 1800],
    "Plant_height": [50, 900, 2400],
    "Grain_weight": [5, 1200, 3500]
}
effect_sizes = [3.5, 5.0, 4.0]  # effect sizes to inject (one per causal SNP)
```

```
# Create simulated phenotypes by adding genetic effects onto real phenotypes
phenotype_sim = {}
for trait in TRAITS:
    base = phenos[trait].astype(float).copy()
    for i, s in enumerate(causal_snps_for_trait.get(trait, [])):
        base = base + G.iloc[:, s] * effect_sizes[i]
    phenotype_sim[trait] = base


def run_gwas(trait_series, G_df):
    results = []

    for snp in G_df.columns:
        X = sm.add_constant(G_df[snp])          # intercept + SNP
        model = sm.OLS(trait_series, X).fit()
        pval = model.pvalues.get(snp, np.nan)
        results.append((snp, pval))
    gwas_df = pd.DataFrame(results, columns=["SNP", "p"])
    gwas_df["-log10(p)"] = -np.log10(gwas_df["p"])
    gwas_df["Pos"] = range(len(gwas_df))
    return gwas_df


all_significant = {}
for trait in TRAITS:
    print(f"\nRunning GWAS for trait: {trait}")
    trait_series = phenotype_sim[trait]
    gwas_df = run_gwas(trait_series, G)

    threshold = -np.log10(0.05 / N_SNPS)
    print(f"Bonferroni threshold (−log10): {threshold:.2f}")

    # Save GWAS table
    gwas_csv = os.path.join(OUT_DIR, f"gwas_{trait}.csv")
    gwas_df.to_csv(gwas_csv, index=False)
    print(f"Saved GWAS table to: {gwas_csv}")

    # Plot Manhattan plot interactively
    plt.figure(figsize=(12,5))
    plt.scatter(gwas_df["Pos"], gwas_df["-log10(p)"], s=10, alpha=0.7)
    plt.axhline(threshold, linestyle="--")
    plt.title(f"Manhattan plot for {trait}")
    plt.xlabel("SNP position")
    plt.ylabel("-log10(p-value)")
    plt.tight_layout()
    plt.show()

    # Save significant SNPs
    sig = gwas_df[gwas_df["-log10(p)"] > threshold].copy().sort_values("-log10(p)",
ascending=False)
    sig_csv = os.path.join(OUT_DIR, f"significant_snps_{trait}.csv")
```

```
        sig.to_csv(sig_csv, index=False)
        all_significant[trait] = sig
        print(f"Significant SNPs: {len(sig)}  (saved to {sig_csv})")


    # Combine top significant SNPs across traits
    combined_list = []
    for trait, df in all_significant.items():
        if not df.empty:
            top = df.head(50).copy()
            top["Trait"] = trait
            combined_list.append(top[["Trait","SNP","p","-log10(p)","Pos"]])
    if combined_list:
        combined_df = pd.concat(combined_list, ignore_index=True)
        combined_csv = os.path.join(OUT_DIR, "combined_significant_snps.csv")
        combined_df.to_csv(combined_csv, index=False)
        print(f"\nCombined significant SNPs saved to: {combined_csv}")
    else:
        print("\nNo significant SNPs across analyzed traits.")

    print(f"\nAll results saved to folder: {OUT_DIR}")

    explain this code li e by line
```

Absolutely, Srushti 🌱 — let's go through your **Python GWAS simulation code line-by-line**, carefully and clearly, so you fully understand what each part is doing.

🧬 Overview

This script simulates **Genome-Wide Association Studies (GWAS)** — a technique in bioinformatics for finding links between genetic variants (SNPs) and traits (like Yield, Plant Height, etc.).

You're using real **phenotype data** (trait measurements from a CSV) and simulating **genotypes (SNPs)**, then performing **OLS regression** per SNP to test its effect on each trait.

🧩 Imports and Setup

```
python

import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import os
```

- **`numpy`** – for numeric operations and random number generation.
- **`pandas`** – for reading/writing CSVs and handling tabular data.
- **`statsmodels.api`** – used for statistical modeling (regression here).

- **`matplotlib.pyplot`** – for plotting graphs like Manhattan plots.
- **`os`** – for file/folder handling (creating directories, joining paths).

🧱 Configuration Section

```
python

PHENOS_PATH = r"C:\Users\srush\Documents\Desktop\cl2\bi\data\phenos.csv"
OUT_DIR = r"C:\Users\srush\Documents\Desktop\cl2\bi\data\Bioinformatics"
N_SNPS = 5000
TRAITS = ["Yield", "Plant_height", "Grain_weight"]
np.random.seed(42)
```

- **`PHENOS_PATH`** → location of your input CSV with real trait (phenotype) values.
- **`OUT_DIR`** → folder where output files (GWAS results, plots) will be saved.
- **`N_SNPS`** → number of SNPs (genetic variants) to simulate.
- **`TRAITS`** → list of traits to analyze.
- **`np.random.seed(42)`** → fixes the random seed for reproducibility (same results each run).

🗂️ Make Output Directory

```
python

os.makedirs(OUT_DIR, exist_ok=True)
```

- Creates the output directory if it doesn't already exist.

📄 Load Phenotype Data

```
python

phenos = pd.read_csv(PHENOS_PATH).reset_index(drop=True)
```

- Reads the phenotype CSV file into a DataFrame named **`phenos`**.
- **`reset_index(drop=True)`** removes any existing index and resets it to 0, 1, 2, ...

✅ Check if Traits Exist

```
python

for t in TRAITS:
    if t not in phenos.columns:
        raise ValueError(f"Trait '{t}' not found in phenos.csv. Columns:
{phenos.columns.tolist()}")
```

- Loops through each trait in **`TRAITS`**.
- If the column doesn't exist in the CSV, it raises an error.
- Prevents proceeding with invalid or missing traits.

🔢 Basic Info

```
python

n_samples = phenos.shape[0]
print(f"Samples: {n_samples}, Traits: {TRAITS}, Simulating {N_SNPS} SNPs...")
```

- Gets number of samples (rows) in your phenotype data.
- Prints info for tracking.

🧬 Simulate SNP Matrix

```
python

G = pd.DataFrame(
    np.random.randint(0, 3, size=(n_samples, N_SNPS)),
    columns=[f"SNP_{i+1}" for i in range(N_SNPS)]
)
```

- Simulates a matrix **`G`** of size **`(n_samples × N_SNPS)`**.
- Each SNP value is 0, 1, or 2 — representing possible genotypes (AA, Aa, aa).
- Column names become "SNP_1", "SNP_2", ... "SNP_5000".

So **`G`** represents all your simulated genetic variants.

🎯 Define Causal SNPs & Effect Sizes

```
python

causal_snps_for_trait = {
    "Yield": [10, 150, 1800],
    "Plant_height": [50, 900, 2400],
    "Grain_weight": [5, 1200, 3500]
}
effect_sizes = [3.5, 5.0, 4.0]
```

- Defines **which SNPs actually affect each trait** (simulated "causal" SNPs).
- **`effect_sizes`** defines how strongly they affect the trait values.

🧪 Create Simulated Phenotypes

```
python

phenotype_sim = {}
for trait in TRAITS:
    base = phenos[trait].astype(float).copy()
    for i, s in enumerate(causal_snps_for_trait.get(trait, [])):
        base = base + G.iloc[:, s] * effect_sizes[i]
    phenotype_sim[trait] = base
```

For each trait:

- Takes the original phenotype values (**`base`**).
- Adds the effect of each causal SNP (genotype × effect size).
- Stores the "genetically influenced" trait in **`phenotype_sim`**.

Essentially, you're injecting true SNP effects into your phenotype data.

🧮 Function: Run GWAS

```
python

def run_gwas(trait_series, G_df):
    results = []

    for snp in G_df.columns:
        X = sm.add_constant(G_df[snp])           # intercept + SNP
        model = sm.OLS(trait_series, X).fit()    # linear regression
        pval = model.pvalues.get(snp, np.nan)    # p-value for SNP effect
        results.append((snp, pval))
    gwas_df = pd.DataFrame(results, columns=["SNP", "p"])
    gwas_df["-log10(p)"] = -np.log10(gwas_df["p"])
    gwas_df["Pos"] = range(len(gwas_df))
    return gwas_df
```

For each SNP:

- Builds a simple linear model:
  Trait = Intercept + β × SNP + Error
- Runs OLS regression using **`statsmodels`**.
- Extracts the **p-value** for that SNP's coefficient.
- Collects all SNPs and their p-values into a DataFrame.
- Adds a new column **`-log10(p)`** for Manhattan plotting.
- Adds a "Pos" index column for plotting along the x-axis.

🧹 Run GWAS for Each Trait

```
python

all_significant = {}
for trait in TRAITS:
    print(f"\nRunning GWAS for trait: {trait}")
```

```
    trait_series = phenotype_sim[trait]
    gwas_df = run_gwas(trait_series, G)
```

Loops through each trait:

- Prints status.
- Gets the simulated phenotype values.
- Runs the GWAS function defined above.

📉 Bonferroni Threshold (Multiple Testing Correction)

```
python

threshold = -np.log10(0.05 / N_SNPS)
print(f"Bonferroni threshold (−log10): {threshold:.2f}")
```

- Corrects for multiple testing (many SNPs = many comparisons).
- Bonferroni correction sets a stricter significance level:
  α = 0.05 / N_SNPS
- Converts that p-value cutoff to **`-log10`** scale for plotting.

💾 Save GWAS Table

```
python

gwas_csv = os.path.join(OUT_DIR, f"gwas_{trait}.csv")
gwas_df.to_csv(gwas_csv, index=False)
print(f"Saved GWAS table to: {gwas_csv}")
```

- Saves the entire GWAS result table (SNPs, p-values, etc.) to CSV.

📊 Manhattan Plot

```
python

plt.figure(figsize=(12,5))
plt.scatter(gwas_df["Pos"], gwas_df["-log10(p)"], s=10, alpha=0.7)
plt.axhline(threshold, linestyle="--")
plt.title(f"Manhattan plot for {trait}")
plt.xlabel("SNP position")
plt.ylabel("-log10(p-value)")
plt.tight_layout()
plt.show()
```

Creates the famous **Manhattan Plot**:

- x-axis → SNP position.
- y-axis → significance (**`-log10(p)`**).

- Horizontal dashed line → significance threshold.
- Visually identifies strong associations (tall peaks).

📄 Save Significant SNPs

```
python

sig = gwas_df[gwas_df["-log10(p)"] > threshold].copy().sort_values("-log10(p)", ascending=False)
sig_csv = os.path.join(OUT_DIR, f"significant_snps_{trait}.csv")
sig.to_csv(sig_csv, index=False)
all_significant[trait] = sig
print(f"Significant SNPs: {len(sig)}  (saved to {sig_csv})")
```

- Filters SNPs that pass the Bonferroni threshold.
- Sorts them by significance.
- Saves to another CSV (only significant ones).
- Keeps track of significant SNPs per trait.

🧩 Combine Top SNPs Across Traits

```
python

combined_list = []
for trait, df in all_significant.items():
    if not df.empty:
        top = df.head(50).copy()
        top["Trait"] = trait
        combined_list.append(top[["Trait","SNP","p","-log10(p)","Pos"]])
```

- From each trait's results, takes top 50 significant SNPs.
- Adds a "Trait" column for reference.
- Stores in a list to merge later.

🧮 Merge and Save Combined Results

```
python

if combined_list:
    combined_df = pd.concat(combined_list, ignore_index=True)
    combined_csv = os.path.join(OUT_DIR, "combined_significant_snps.csv")
    combined_df.to_csv(combined_csv, index=False)
    print(f"\nCombined significant SNPs saved to: {combined_csv}")
else:
    print("\nNo significant SNPs across analyzed traits.")
```

- Combines all top SNPs into a single CSV for easy comparison.
- Handles case where no SNPs were significant.

✅ Final Output Message

```
python

print(f"\nAll results saved to folder: {OUT_DIR}")
```

- Finishing message summarizing that all results are saved.

🧠 Summary of What the Script Does

Step	Description
1️⃣	Load real phenotype data
2️⃣	Simulate 5000 random SNPs
3️⃣	Add true genetic effects to phenotypes
4️⃣	Run OLS regression for each SNP vs. each trait
5️⃣	Apply Bonferroni correction
6️⃣	Save results and plot Manhattan plots
7️⃣	Save significant SNPs and merge across traits

Would you like me to **add comments directly inside your code** (turning it into a fully explained, annotated version you can run and read together)?

> what is snp in short

In short —

🧬 **SNP** stands for **Single Nucleotide Polymorphism**.

It's a **small genetic variation** where **one nucleotide (A, T, C, or G)** in the DNA sequence differs among individuals.

For example:

```
less

Person 1: A T C G A
Person 2: A T T G A
                ↑
            (SNP difference)
```

In simple terms:

- SNP = one-letter change in DNA.
- These tiny changes can affect how genes work and influence traits (like height, yield, or disease resistance).

So, in GWAS, we test **each SNP** to see if it's **statistically associated** with a specific trait.