QTL Analysis Pipeline - Complete Documentation

Table of Contents

Overview

What is QTL Analysis?

Quantitative Trait Locus (QTL) analysis identifies genomic regions where genetic variation is associated with variation in molecular traits. This pipeline provides a comprehensive solution for three main types of QTL analyses:

- eQTL Analysis: Identifies genetic variants that influence gene expression levels
- pQTL Analysis: Identifies genetic variants that affect protein abundance
- sQTL Analysis: Identifies genetic variants that impact RNA splicing patterns
- GWAS Analysis: Optional genome-wide association study for complex traits

Pipeline Features

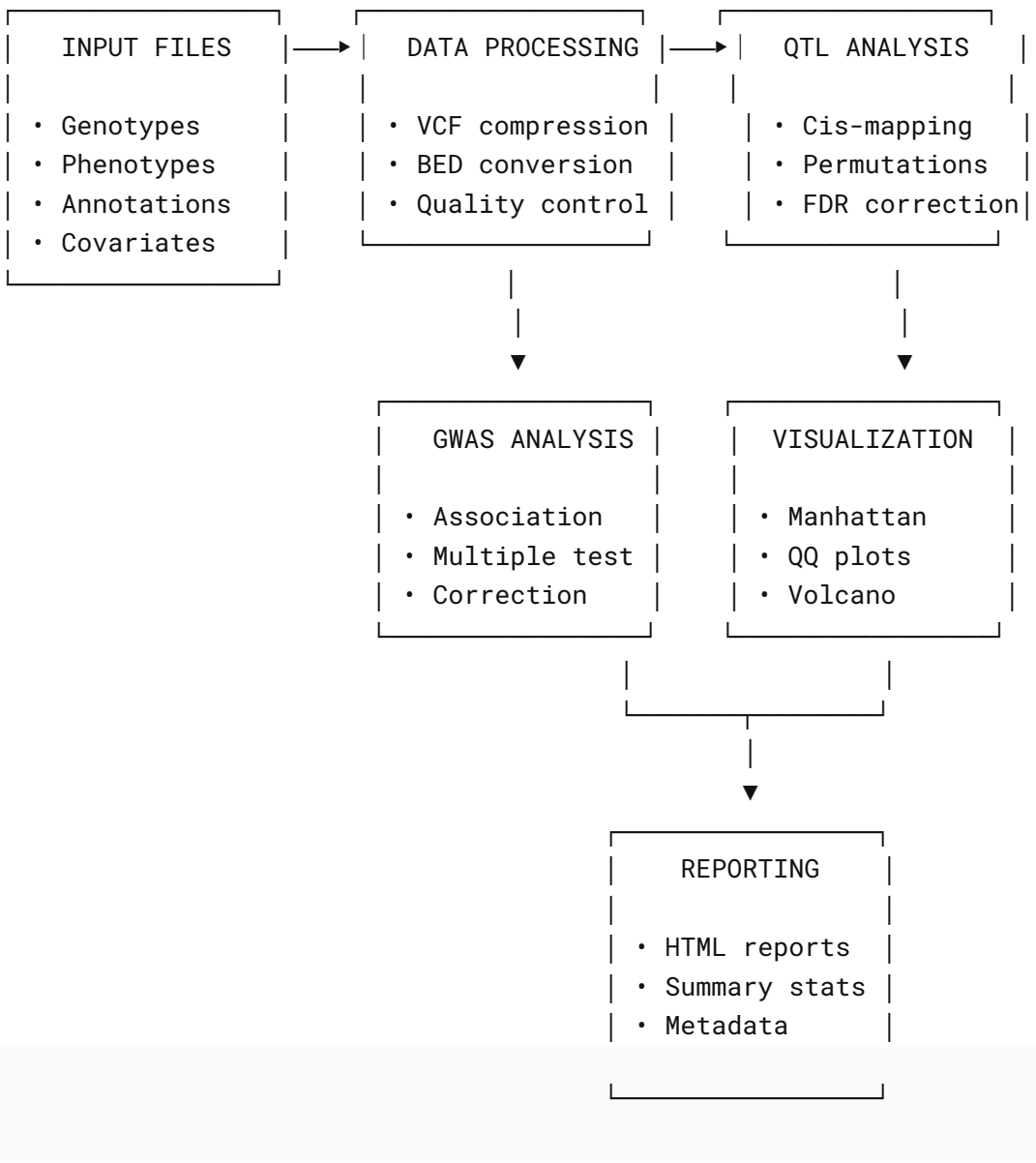
- Comprehensive Analysis: Supports multiple QTL types and optional GWAS
- Professional Visualization: Generates publication-ready plots
- Robust Quality Control: Comprehensive input validation and error handling

- Flexible Configuration: YAML-based configuration system
- Detailed Reporting: HTML and text reports with results summary
- Production Ready: Proper logging, error handling, and output organization

Pipeline Architecture

High-Level Workflow

text



Core Components

1. Configuration System

- File: `config/config.yaml`
- Purpose: Centralized configuration management
- Features: YAML format with mandatory/optional parameters

2. Input Validation

- Script: `scripts/utils/validation.py`
- Purpose: Validate all input files and data consistency
- Checks: File existence, format, sample concordance, data integrity

3. Genotype Processing

- Script: `scripts/utils/qtl_analysis.py`
- Purpose: Prepare genotype data for analysis
- Functions: VCF compression, indexing, format conversion

4. Phenotype Processing

- Script: `scripts/utils/qtl_analysis.py`
- Purpose: Convert phenotype data to BED format
- Functions: Annotation mapping, BED creation, compression

5. QTL Analysis

- Script: `scripts/utils/qtl_analysis.py`
- Purpose: Perform QTL mapping using QTLTools
- Methods: Cis-window mapping, permutation testing, FDR correction

6. GWAS Analysis

- Script: `scripts/utils/gwas_analysis.py`
- Purpose: Perform genome-wide association studies
- Methods: Linear/logistic regression, multiple testing correction

7. Visualization

- Script: `scripts/utils/plotting.py`

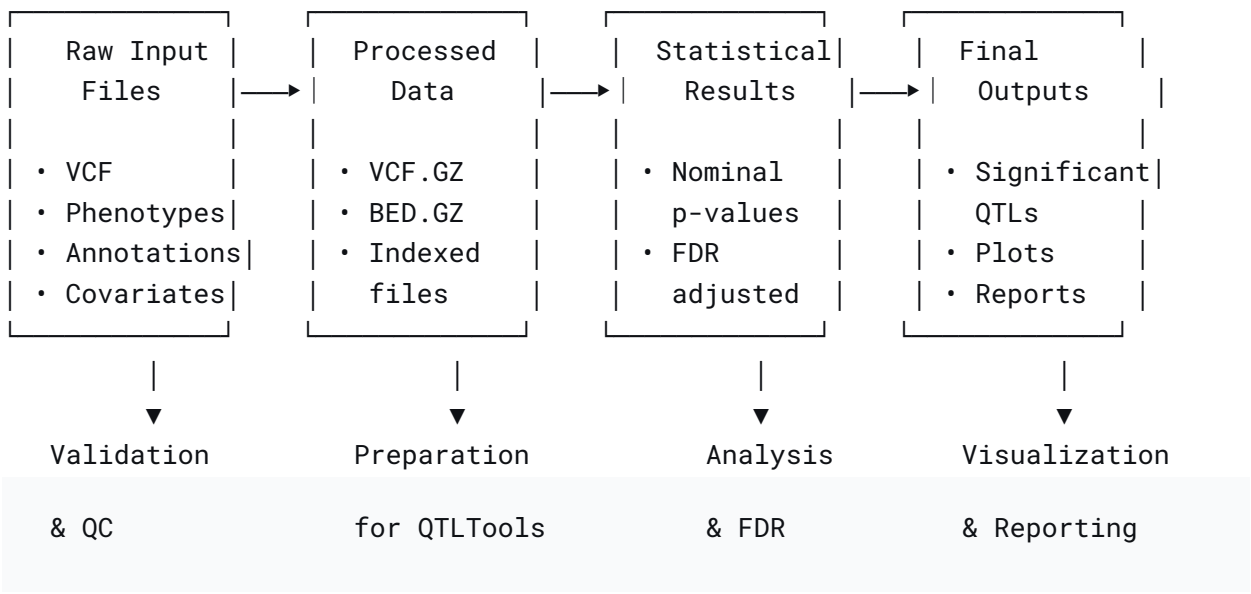
- Purpose: Generate comprehensive plots
- Plot Types: Manhattan, QQ, volcano, distribution, summary plots

8. Reporting

- Script: `scripts/utils/report_generator.py`
- Purpose: Generate analysis reports
- Outputs: HTML reports, text summaries, metadata

Data Flow Diagram

```
text
```



Installation & Setup

System Requirements

- Operating System: Linux or macOS
- Memory: 8GB minimum, 16GB+ recommended
- Storage: 20GB+ free space
- Python: Version 3.7 or higher

Required Tools Installation

1. Install Bioinformatics Tools

```bash
# Using conda (recommended)
conda install -c bioconda qtltools plink bcftools htslib

# Or install individually:
# QTLTools: https://qtltools.github.io/qtltools/
# PLINK: https://www.cog-genomics.org/plink/
# BCFtools: http://www.htslib.org/
```

2. Install Python Dependencies

```bash
pip install pandas numpy matplotlib seaborn scipy pyyaml
```

3. Verify Installation

```bash
# Check tool availability
qtltools --version
plink --version
bcftools --version

python -c "import pandas, numpy, matplotlib; print('Python packages installed')"
```

Pipeline Setup

```bash
# Download and setup the pipeline
git clone https://github.com/SINGHVJ-Bio/QTL_ANALYSIS.git
cd QTL_ANALYSIS
```

```
# Make scripts executable
chmod +x run_QTLPipeline.py
chmod +x scripts/*.py scripts/utils/*.py

# Test installation
python run_QTLPipeline.py --config config/config.yaml --validate-only
```

Input Requirements

Mandatory Input Files

1. Genotype File (VCF/VCF.GZ)

Format: VCF (Variant Call Format) or compressed VCF.GZ
Requirements:

- Must contain all samples used in the analysis
- Should include chromosome, position, and genotype information
- Can be whole genome or targeted sequencing data

Example VCF structure:

```
text
#CHROM POS ID REF ALT QUAL FILTER INFO FORMAT sample1 sample2 sample3
chr1   1000 rs1 A G . PASS . GT 0/0 0/1 1/1
chr1   2000 rs2 C T . PASS . GT 0/1 0/0 0/1
```

2. Covariates File

Format: Tab-separated values (TSV)
Structure:

- Rows: Covariates (PCs, sex, age, batch effects)
- Columns: Sample IDs
- First column: Covariate names

Example covariates.txt:

```
text
```

```
ID      sample1 sample2 sample3 sample4
PC1     0.01    -0.02  0.03    -0.01
PC2     -0.03   0.01   -0.02   0.04
```

```
sex     1       2      1       2
```

3. Annotation File (BED Format)

Format: BED (Browser Extensible Data)
Purpose: Maps molecular features to genomic coordinates

Example annotations.bed:

```
text
```

```
#chr   start   end     gene_id     strand
chr1   1000000 1005000 ENSG000001  +
chr1   2000000 2008000 ENSG000002  -
```

```
chr1   3000000 3010000 ENSG000003  +
```

4. Phenotype Files

Format: Tab-separated values (TSV)
Structure:

- Rows: Molecular features (genes, proteins, splicing events)
- Columns: Sample IDs
- First column: Feature IDs

Example expression.txt:

```
text
```

```
GeneID      sample1 sample2 sample3 sample4
ENSG000001  10.5    11.2    9.8     12.1
ENSG000002  8.3     7.9     9.1     8.7
```

```
ENSG000003  15.2    14.8    16.1    15.7
```

Optional Input Files

GWAS Phenotype File

Required only if running GWAS analysis
Format: Tab-separated values with sample_id column

Example gwas_phenotype.txt:

```
text
```

```
sample_id    height    weight    disease_status
sample1     175.2     68.5      0
sample2     168.7     72.1      1
sample3     182.3     75.8      0
```

Configuration Guide

Configuration File Structure

The pipeline uses `config/config.yaml` for all settings:

```
yaml
```

```yaml
# MANDATORY: Results directory where all output will be stored
results_dir: "results"

# MANDATORY: Input files section
input_files:
  genotypes: "data/genotypes.vcf"        # VCF or VCF.GZ file
  covariates: "data/covariates.txt"      # Covariates file
  annotations: "data/annotations.bed"    # BED annotations
  expression: "data/expression.txt"      # For eQTL analysis
  protein: "data/protein.txt"            # For pQTL analysis
  splicing: "data/splicing.txt"          # For sQTL analysis

# OPTIONAL: Analysis configuration
analysis:
```

```yaml
  qtl_types: "all"              # "all", "eqtl", "pqtl", "sqtl" or
comma-separated
  run_gwas: false               # Enable GWAS analysis
  gwas_phenotype: "data/gwas_phenotype.txt"  # Required if run_gwas is true

# OPTIONAL: QTL analysis parameters
qtl:
  cis_window: 1000000           # Cis window size in base pairs
  permutations: 1000            # Number of permutations for FDR
  maf_threshold: 0.05           # Minor allele frequency threshold
  fdr_threshold: 0.05           # FDR threshold for significance
  min_maf: 0.01                 # Minimum MAF for variants
  min_call_rate: 0.95           # Minimum call rate

# OPTIONAL: GWAS parameters
gwas:
  method: "linear"              # "linear" or "logistic"
  covariates: true              # Adjust for covariates
  maf_threshold: 0.01           # MAF filter for GWAS
  imputation: "mean"            # Missing data handling

# OPTIONAL: Plotting configuration
plotting:
  enabled: true                 # Enable/disable plotting
  dpi: 300                      # Plot resolution
  format: "png"                 # "png", "pdf", or "svg"
  style: "seaborn"              # Plot style
  plot_types:                   # Types of plots to generate
    - "manhattan"
    - "qq"
    - "volcano"
    - "distribution"
    - "summary"

# OPTIONAL: Output settings
output:
  compression: true             # Compress intermediate files
  remove_intermediate: false    # Remove temp files to save space
  generate_report: true         # Generate HTML report
  report_format: "html"         # Report format

# OPTIONAL: Quality control
```

```yaml
qc:
  check_sample_concordance: true
  filter_low_expressed: true
  expression_threshold: 0.1
  normalize: true

# OPTIONAL: Tool paths (if not in system PATH)
paths:
  qtltools: "qtltools"
  bcftools: "bcftools"
  bgzip: "bgzip"
  tabix: "tabix"
  python: "python3"

  plink: "plink"
```

Key Configuration Parameters

Analysis Types

- qtl_types: Specify which QTL analyses to run
  - `"all"`: Run all available QTL analyses
  - `"eqtl"`: Run only eQTL analysis
  - `"eqtl,pqtl"`: Run both eQTL and pQTL analyses

Statistical Parameters

- cis_window: Distance from gene to consider variants (default: 1Mb)
- permutations: Number of permutations for empirical p-values (default: 1000)
- fdr_threshold: False Discovery Rate threshold for significance (default: 0.05)

Quality Control

- maf_threshold: Exclude variants with MAF below this threshold
- min_call_rate: Exclude variants with high missingness
- expression_threshold: Filter lowly expressed genes

Running the Pipeline

Basic Usage

1. Run Complete Analysis

```bash
python run_QTLPipeline.py --config config/config.yaml
```

2. Run Specific QTL Types

```bash
# Run only eQTL and pQTL analyses
python run_QTLPipeline.py --config config/config.yaml --analysis-types
eqtl,pqtl

# Run only eQTL analysis
python run_QTLPipeline.py --config config/config.yaml --analysis-types eqtl
```

3. Run with GWAS Analysis

```bash
python run_QTLPipeline.py --config config/config.yaml --run-gwas
```

4. Validate Inputs Only

```bash
python run_QTLPipeline.py --config config/config.yaml --validate-only
```

Using Shell Wrapper

```bash
# Make shell script executable
chmod +x shellScript/run_qtl_pipeline.sh

# Run using shell wrapper
```

```
./shellScript/run_qtl_pipeline.sh -c config/config.yaml -a eqtl,pqtl -g
```

Command Line Options

Option	Description	Default
`--config`	Path to configuration file (required)	-
`--analysis-types`	Override QTL types from config	config value
`--run-gwas`	Enable GWAS analysis	false
`--validate-only`	Only validate inputs, don't run analysis	false

Example Workflows

Workflow 1: Complete Multi-omics Analysis

```
bash
# Configure config.yaml with all phenotype files
# Then run:
python run_QTLPipeline.py --config config/config.yaml --analysis-types all
--run-gwas
```

Workflow 2: Expression-focused Analysis

```
bash
# Only run eQTL analysis with detailed plotting
python run_QTLPipeline.py --config config/config.yaml --analysis-types eqtl
```

Workflow 3: Protein QTL Analysis

```bash
# Focus on protein QTLs with custom parameters
python run_QTLPipeline.py --config config/config.yaml --analysis-types pqtl
```

Output Structure

Directory Organization

```text
results/
├── qtl_results/                # Primary QTL results
│   ├── eqtl_significant.txt    # FDR-significant eQTL associations
│   ├── eqtl_nominals.txt       # All nominal eQTL associations
│   ├── pqtl_significant.txt    # FDR-significant pQTL associations
│   ├── pqtl_nominals.txt       # All nominal pQTL associations
│   ├── sqtl_significant.txt    # FDR-significant sQTL associations
│   └── sqtl_nominals.txt       # All nominal sQTL associations
├── gwas_results/               # GWAS results (if enabled)
│   ├── gwas_combined_results.txt      # Combined GWAS results
│   ├── gwas_phenotype1.assoc.linear   # Per-phenotype results
│   └── gwas_phenotype2.assoc.linear
├── plots/                      # Visualization outputs
│   ├── eqtl_manhattan.png      # eQTL Manhattan plot
│   ├── eqtl_qq.png             # eQTL Q-Q plot
│   ├── eqtl_volcano.png        # eQTL volcano plot
│   ├── eqtl_distribution.png   # eQTL p-value distribution
│   ├── pqtl_manhattan.png      # pQTL Manhattan plot
│   ├── sqtl_manhattan.png      # sQTL Manhattan plot
│   ├── gwas_manhattan.png      # GWAS Manhattan plot
│   ├── gwas_qq.png             # GWAS Q-Q plot
│   └── analysis_summary.png    # Cross-analysis summary
├── reports/                    # Comprehensive reports
│   ├── analysis_report.html    # Interactive HTML report
│   ├── pipeline_summary.txt    # Text summary of results
│   └── results_metadata.json   # Analysis metadata and parameters
├── logs/                       # Execution logs
│   └── pipeline_20240115_143022.log  # Timestamped log file
└── temp/                       # Intermediate files (optional)
```

```
    └── genotypes/
        ├── genotypes.vcf.gz   # Compressed genotype file
        └── genotypes.vcf.gz.tbi  # Tabix index
```

File Formats and Contents

Significant QTL Results

File: `qtl_results/[type]_significant.txt`
Format: Tab-separated values
Columns:

- `feature_id`: Molecular feature ID (e.g., gene ID)
- `variant_id`: Genetic variant ID (e.g., chr1_1000000)
- `p_value`: Nominal p-value from association test
- `beta`: Effect size estimate
- `p_adjusted`: FDR-adjusted p-value
- `p_threshold`: Significance threshold used

Example:

```text
feature_id    variant_id      p_value    beta    p_adjusted
ENSG000001    chr1_1000000    1.2e-08    0.45    0.001
ENSG000001    chr1_1500000    3.4e-07   -0.32    0.015
ENSG000002    chr2_2000000    2.1e-06    0.28    0.042
```

Nominal QTL Results

File: `qtl_results/[type]_nominals.txt`
Format: Tab-separated values with all tested associations
Contents: All variant-feature pairs tested, including non-significant results

GWAS Results

File: `gwas_results/gwas_combined_results.txt`
Format: Tab-separated values
Columns:

- `CHR`: Chromosome
- `SNP`: Variant identifier
- `BP`: Base pair position
- `A1`: Effect allele
- `A2`: Other allele
- `BETA`: Effect size estimate
- `SE`: Standard error of effect size
- `P`: P-value from association test
- `PHENOTYPE`: Phenotype name

Plot Descriptions

Manhattan Plots

- Purpose: Visualize genome-wide association signals
- Interpretation:
    - Each point represents a genetic variant
    - X-axis: Genomic position
    - Y-axis: -log10(p-value)
    - Red line: Genome-wide significance threshold ($p < 5 \times 10^{-8}$)
    - Orange line: Suggestive significance threshold ($p < 1 \times 10^{-5}$)

Q-Q Plots

- Purpose: Assess inflation of test statistics
- Interpretation:
    - X-axis: Expected -log10(p-values) under null hypothesis
    - Y-axis: Observed -log10(p-values)
    - Diagonal line: Expected under no inflation
    - λ (lambda): Genomic control coefficient (λ > 1 indicates inflation)

Volcano Plots

- Purpose: Visualize effect size vs statistical significance
- Interpretation:

    - X-axis: Effect size (beta)
    - Y-axis: -log10(p-value)
    - Points above horizontal line: Statistically significant
    - Points beyond vertical lines: Large effect sizes

Distribution Plots

- Purpose: Show distribution of p-values and effect sizes
- Interpretation:
    - Left panel: P-value distribution (should be uniform under null)
    - Right panel: Effect size distribution

Summary Plots

- Purpose: Compare results across different analysis types
- Interpretation: Bar plot showing number of significant associations for each analysis

Results Interpretation

Understanding QTL Results

Statistical Significance

- Nominal p-value: Raw p-value from association test
- FDR-adjusted p-value: Corrected for multiple testing using False Discovery Rate
- Significance threshold: Typically FDR < 0.05

Effect Sizes

- Beta coefficient: Effect size estimate
- Positive beta: Variant associated with increased trait levels
- Negative beta: Variant associated with decreased trait levels
- Magnitude: Strength of association (larger absolute value = stronger effect)

Biological Interpretation

eQTL Results

- Cis-eQTL: Variants near the gene they regulate (within 1Mb)
- Trans-eQTL: Variants distant from the regulated gene
- Interpretation: Genetic variant influences gene expression levels

pQTL Results

- Interpretation: Genetic variant influences protein abundance
- Importance: May have direct functional consequences

sQTL Results

- Interpretation: Genetic variant influences RNA splicing patterns
- Relevance: Can affect protein function through alternative splicing

Quality Assessment

Q-Q Plots

- $\lambda \approx 1$: Minimal inflation, good quality control
- $\lambda > 1.05$: Possible population stratification or technical artifacts
- $\lambda < 1$: Over-correction or conservative testing

Manhattan Plots

- Good pattern: Few peaks above significance threshold
- Concerning: Many peaks across genome (possible stratification)
- Expected: Most variants show no association (null distribution)

Multiple Testing Considerations

The pipeline addresses multiple testing through:

1. Cis-window restriction: Only testing variants near genes
2. Permutation testing: Empirical null distribution
3. FDR correction: Control of false discoveries

Script Documentation

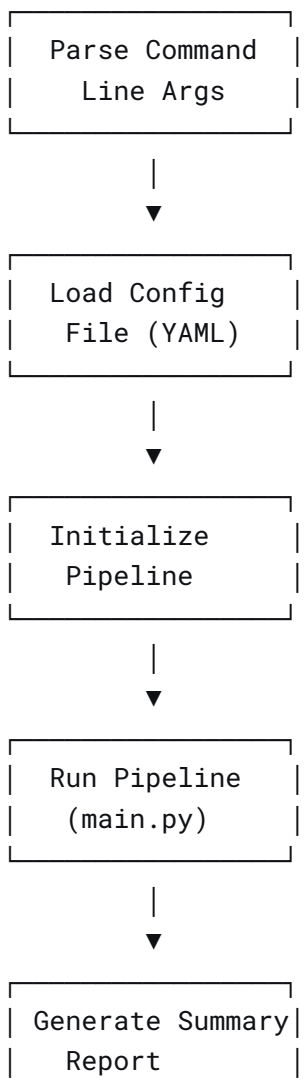
Main Runner Script: `run_QTLPipeline.py`

Purpose

Main entry point for the QTL analysis pipeline, located in root directory for easy access.

Flow Diagram

```
text
```



```
└──────────────────┘
```

Key Functions

- `main()`: Parse arguments and initialize pipeline
- Command line interface handling
- Results summary generation

Usage

```bash
python run_QTLPipeline.py --config config.yaml [options]
```

Pipeline Orchestrator: `scripts/main.py`

Purpose

Orchestrates the complete QTL analysis workflow and coordinates all components.

Flow Diagram

```text
┌──────────────────┐    ┌──────────────────┐    ┌──────────────────┐
│   Input          │    │  Genotype        │    │   QTL Analysis   │
│  Validation      │──→ │   Processing     │──→ │    (QTLTools)    │
└──────────────────┘    └──────────────────┘    └──────────────────┘
         │                       │                       │
         ▼                       ▼                       ▼
┌──────────────────┐    ┌──────────────────┐    ┌──────────────────┐
│  GWAS Analysis   │    │   Plotting       │    │    Reporting     │
│    (PLINK)       │    │  (Matplotlib)    │    │   (HTML/Text)    │
└──────────────────┘    └──────────────────┘    └──────────────────┘
```

Class: QTLPipeline

Methods:

- `__init__(config_file)`: Initialize pipeline with configuration
- `setup_directories()`: Create output directory structure
- `setup_logging()`: Configure logging system
- `run_pipeline()`: Execute complete analysis workflow
- `run_qtl_analyses(vcf_gz)`: Run specified QTL analyses
- `run_gwas_analysis(vcf_gz)`: Run GWAS analysis if enabled
- `generate_plots()`: Create all requested visualizations
- `generate_reports()`: Generate comprehensive reports

Key Features

- Comprehensive error handling and logging
- Parallel execution of independent analyses
- Progress tracking and status updates
- Resource management and cleanup

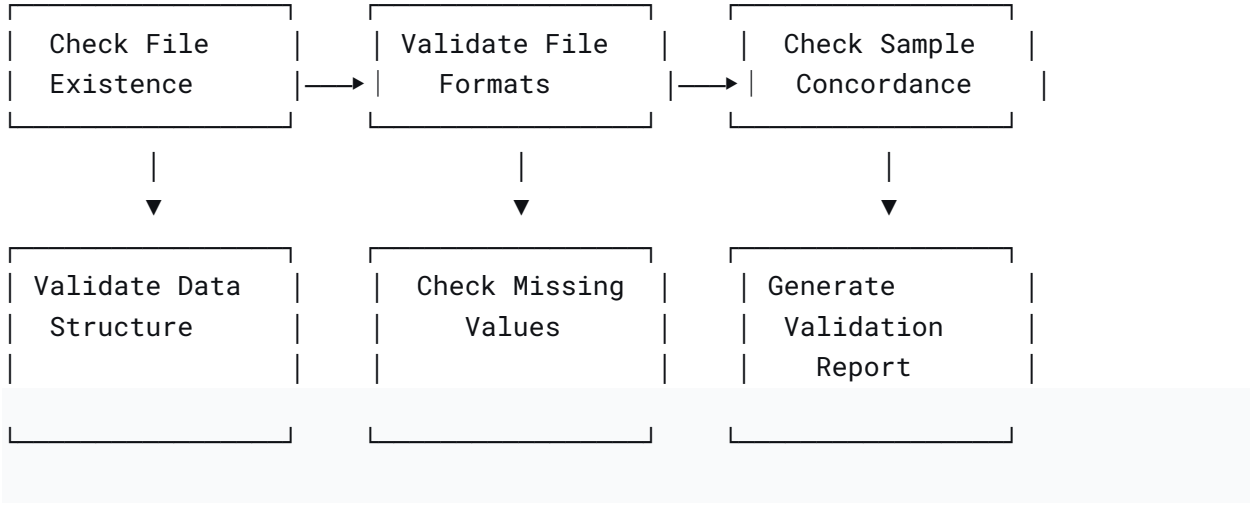
Input Validation: `scripts/utils/validation.py`

Purpose

Validate all input files for format, consistency, and completeness.

Flow Diagram

```
text
```



Functions

- `validate_inputs(config)`: Main validation function
- `validate_phenotype_file(file_path, qtl_type, warnings)`: Check phenotype files
- `validate_gwas_phenotype_file(file_path, warnings)`: Validate GWAS phenotypes
- `validate_file_formats(input_files, warnings)`: Check file formats

Validation Checks

1. File Existence: All required files present
2. Format Compliance: Correct file formats (VCF, BED, TSV)
3. Sample Concordance: Consistent samples across files
4. Data Integrity: No excessive missing values or constant features
5. Annotation Mapping: Features in phenotype files have genomic coordinates

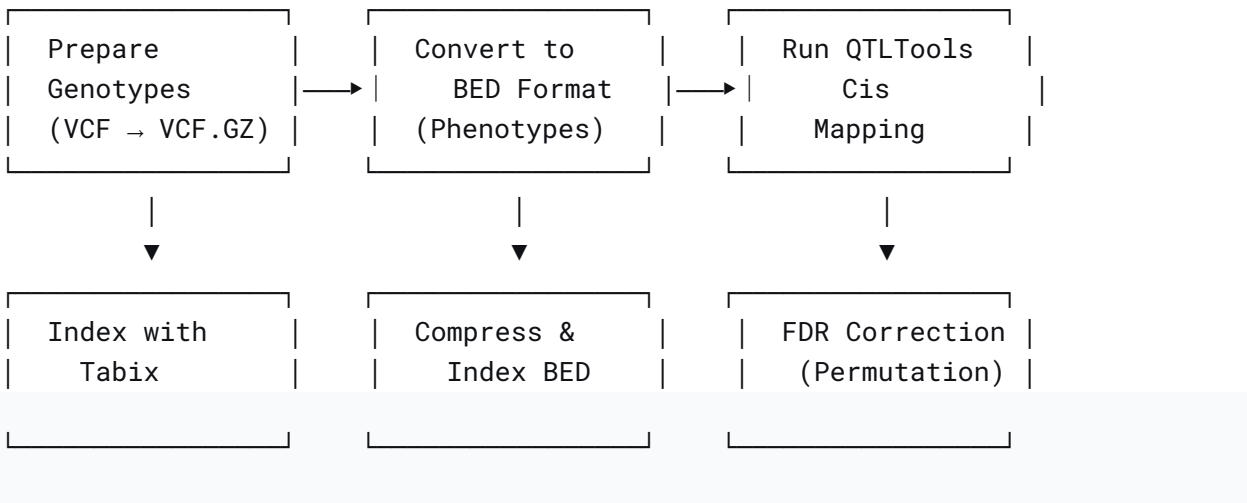
QTL Analysis: `scripts/utils/qtl_analysis.py`

Purpose

Perform QTL mapping analysis using QTLTools for eQTL, pQTL, and sQTL analyses.

Flow Diagram

```
text
```



Functions

- `prepare_genotypes(config, results_dir)`: Process genotype data
- `prepare_phenotype_data(config, qtl_type, results_dir)`: Convert phenotypes to BED
- `run_qtl_analysis(config, vcf_gz, qtl_type, results_dir)`: Run QTL mapping
- `run_command(cmd, description, config)`: Execute shell commands

QTLTools Commands

1. Cis Mapping:
2. bash

```
qtltools cis --vcf genotypes.vcf.gz --bed phenotypes.bed.gz \
             --cov covariates.txt --window 1000000 \
```

3. `             --permute 1000 --out nominals.txt`
4. FDR Correction:
5. bash

```
qtltools correct --qtl nominals.txt --out significant.txt \
```

6. `                 --threshold 0.05`

Statistical Methods

- Association testing: Linear regression
- Multiple testing: Permutation-based FDR
- Cis window: Default 1Mb from gene boundaries
- Covariate adjustment: Principal components and other confounders

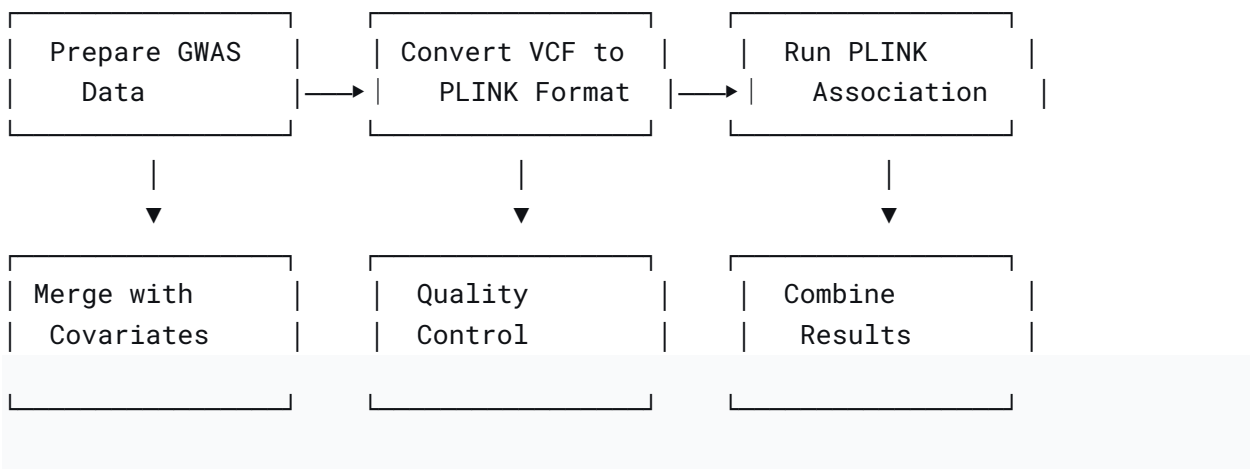
GWAS Analysis: `scripts/utils/gwas_analysis.py`

Purpose

Perform genome-wide association studies using PLINK for complex traits.

Flow Diagram

```
text
```



Functions

- `run_gwas_analysis(config, vcf_gz, results_dir)`: Main GWAS function
- `prepare_gwas_data(config, results_dir)`: Prepare phenotype and covariate data
- `run_plink_gwas(config, vcf_gz, gwas_data, results_dir)`: Run PLINK analysis
- `count_significant_gwas(result_file, pval_threshold)`: Count significant hits

PLINK Commands

1. Format Conversion:
2. ```
   bash
   ```
3. ```
   plink --vcf genotypes.vcf.gz --make-bed --out genotypes
   ```
4. Linear Regression:
5. ```
   bash
   ```

```
plink --bfile genotypes --pheno phenotype.txt --linear \
```

6. ```
   --covar covariates.txt --out gwas_results
   ```

GWAS Methods

- Association model: Linear or logistic regression
- Covariate adjustment: Yes (configurable)
- Quality control: MAF filtering, missingness checks

- Multiple testing: Genome-wide significance ($p < 5 \times 10^{-8}$)

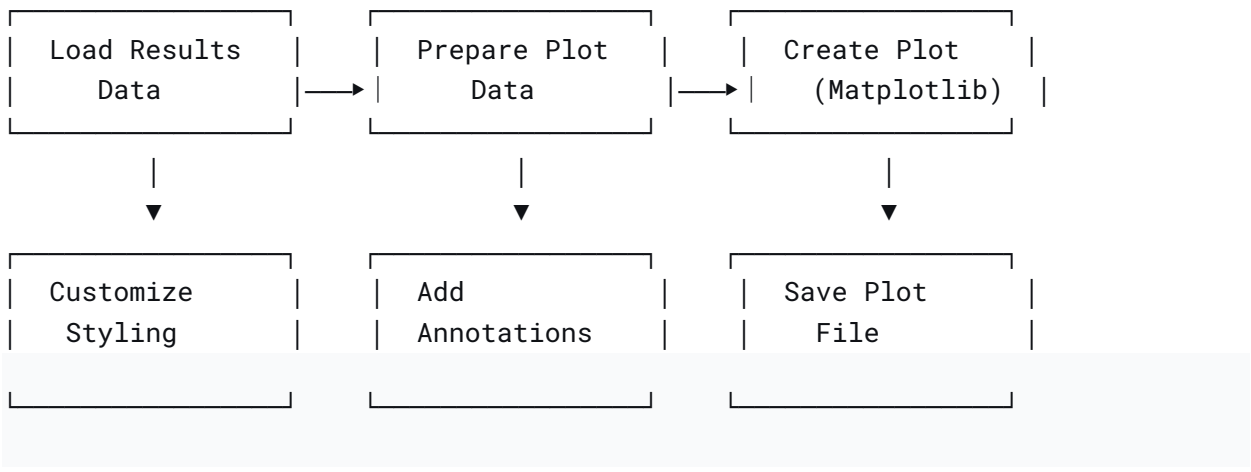
Plotting Utilities: `scripts/utils/plotting.py`

Purpose

Generate comprehensive, publication-ready visualizations for QTL and GWAS results.

Flow Diagram

```
text
```



Class: QTLPlotter

Methods:

- `__init__(config, results, plots_dir)`: Initialize plotter
- `create_qtl_plots(qtl_type, result)`: Create all plots for a QTL type
- `create_gwas_plots(gwas_result)`: Create GWAS plots
- `create_summary_plots()`: Create cross-analysis summary plots
- Individual plot methods: `create_*_manhattan()`, `create_*_qq()`, etc.

Plot Types

1. Manhattan Plots: Genome-wide association signals
2. Q-Q Plots: P-value distribution assessment
3. Volcano Plots: Effect size vs significance
4. Distribution Plots: P-value and effect size distributions

5. Summary Plots: Cross-analysis comparisons

Customization Options

- Color schemes and styles
- Plot dimensions and resolution
- Output formats (PNG, PDF, SVG)
- Significance thresholds and annotations

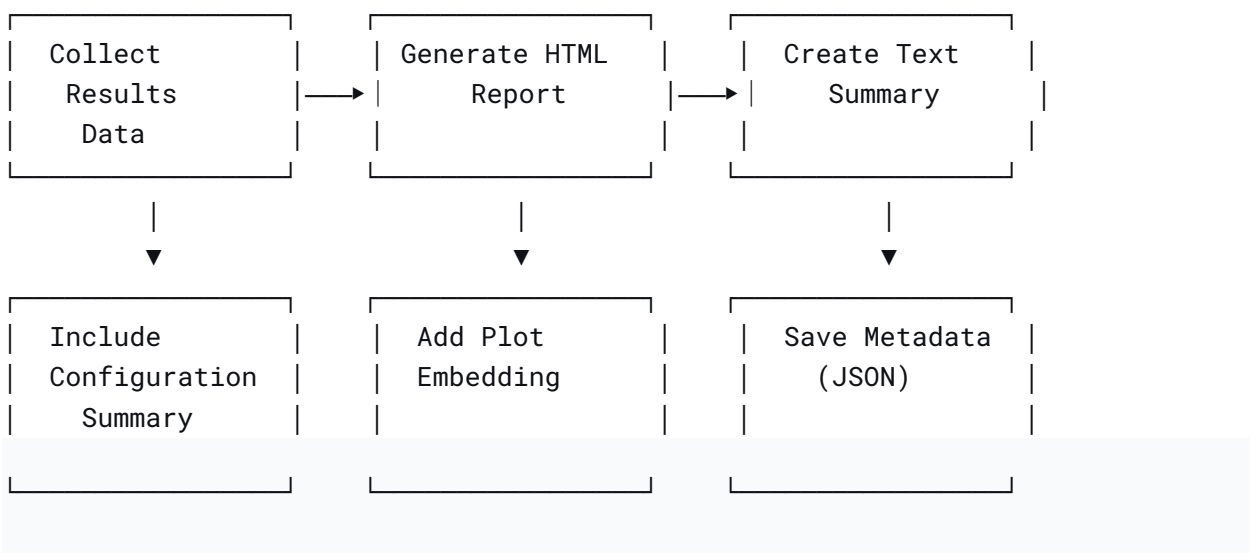
Report Generator: `scripts/utils/report_generator.py`

Purpose

Generate comprehensive HTML and text reports summarizing analysis results.

Flow Diagram

```
text
```



Functions

- `generate_html_report(report_data, output_file)`: Create HTML report
- `generate_summary_report(report_data, output_file)`: Create text summary
- `generate_analysis_section(report_data)`: Generate results section
- `generate_plot_section(report_data)`: Create plot embedding section
- `generate_config_section(report_data)`: Generate configuration summary

Report Features

- Interactive HTML: Navigable sections with embedded plots
- Summary Statistics: Counts of significant associations
- Configuration Summary: Analysis parameters and settings
- Quality Metrics: Lambda GC, sample sizes, other QC metrics
- Plot Gallery: All generated visualizations

Troubleshooting

Common Issues and Solutions

1. File Not Found Errors

Problem: Pipeline cannot find input files
Solution:

```
bash
```

```
# Check file paths in config.yaml
# Use absolute paths for clarity
# Verify file permissions

ls -la /path/to/your/file.vcf
```

2. Tool Not Found Errors

Problem: QTLTools, PLINK, or other tools not found
Solution:

```
bash
```

```
# Check if tools are in PATH
which qtltools
which plink

# Install missing tools
conda install -c bioconda qtltools plink bcftools

# Or add to PATH
```

```
export PATH=/path/to/tools:$PATH
```

3. Memory Issues

Problem: Pipeline runs out of memory
Solution:

- Reduce number of permutations in config
- Use subset of data for testing
- Increase system RAM
- Close other memory-intensive applications

4. Permission Errors

Problem: Cannot write to output directory
Solution:

```
bash
```

```
# Check and set permissions
chmod +x run_QTLPipeline.py
chmod +x scripts/*.py scripts/utils/*.py

# Ensure write access to results directory
mkdir -p results
```

```
chmod 755 results
```

5. Python Package Errors

Problem: Missing Python dependencies
Solution:

```
bash
```

```
# Install required packages
pip install --upgrade pandas numpy matplotlib seaborn scipy pyyaml

# Or using conda
```

```
conda install pandas numpy matplotlib seaborn scipy pyyaml
```

Debugging Steps

1. Validate Inputs

```bash
python run_QTLPipeline.py --config config.yaml --validate-only
```

2. Check Log Files

```bash
# Examine the detailed log file
tail -f results/logs/pipeline_*.log
```

3. Test with Example Data

```bash
# Run with small example dataset first
python run_QTLPipeline.py --config config/config.yaml --analysis-types eqtl
```

4. Check Intermediate Files

```bash
# Verify intermediate processing steps
ls -la results/temp/genotypes/
ls -la results/qtl_results/
```

Performance Optimization

For Large Datasets

1. Use VCF.GZ instead of VCF
2. Increase memory allocation
3. Use fewer permutations for initial runs

4. Process chromosomes separately
5. Use high-performance computing cluster

Memory Usage Tips

- Monitor memory during execution
- Use `--remove-intermediate true` to save space
- Compress intermediate files
- Process data in chunks if possible

Advanced Usage

Custom Analysis Configurations

1. Tissue-Specific QTL Analysis

```yaml
analysis:
  qtl_types: "eqtl"

qtl:
  cis_window: 500000  # Smaller window for tissue-specific analysis
  permutations: 10000 # More permutations for power
  fdr_threshold: 0.01 # Stricter FDR threshold

plotting:
  plot_types: ["manhattan", "qq", "volcano"]
```

2. Multi-omics Integration

```yaml
analysis:
  qtl_types: "eqtl,pqtl,sqtl"
  run_gwas: true

output:
  generate_report: true
```

```yaml
  report_format: "html"

plotting:
  enabled: true
  plot_types: ["manhattan", "qq", "summary"]
```

3. Fine-Mapping Configuration

```yaml
qtl:
  cis_window: 1000000   # Standard cis window
  permutations: 10000   # High permutations for fine-mapping
  maf_threshold: 0.01   # Include rare variants
  fdr_threshold: 0.05   # Standard FDR

gwas:
  method: "linear"
  maf_threshold: 0.01   # Include low-frequency variants
  covariates: true
```

Extending the Pipeline

Adding New QTL Types

1. Add new phenotype file path to config
2. Update validation in `validation.py`
3. Add plotting support in `plotting.py`
4. Update report templates

Custom Plotting

```python
# Example: Add custom plot type
def create_custom_heatmap(self, qtl_type, result):
    # Implementation for custom heatmap
    pass
```

Integration with Other Tools

The modular design allows integration with:

- COLOC: Colocalization analysis
- SMR: Summary-data-based Mendelian Randomization
- FUMA: Functional mapping and annotation
- LocusZoom: Regional association plots

Best Practices

1. Data Preparation

- Quality control genotypes and phenotypes before analysis
- Check for batch effects and population stratification
- Normalize phenotype data appropriately
- Ensure sample concordance across files

2. Analysis Strategy

- Start with small test runs to verify setup
- Use appropriate cis-window sizes for your study
- Consider biological context when interpreting results
- Validate findings in independent datasets when possible

3. Result Interpretation

- Consider both statistical significance and effect sizes
- Account for multiple testing appropriately
- Interpret results in biological context
- Consider functional validation of key findings

4. Reproducibility

- Version control for configuration files
- Document all parameter choices
- Save complete analysis logs
- Use containerization for computational environment