**main.py**

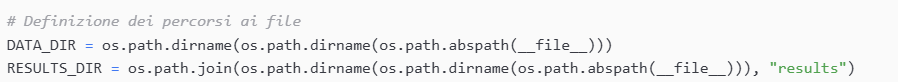
main.py è lo script principale del progetto che prende in input un gene da analizzare, il file FASTA della sua sequenza e un file CSV di varianti. Utilizza moduli esterni per caricare sequenze e varianti, filtrare le varianti per il gene di interesse, calcolare statistiche di base e generare visualizzazioni e un report di riepilogo, organizzando i risultati in directory specifiche.

**1. Intestazione del file e importazioni**

A screen shot of a computer

AI-generated content may be incorrect.

* La prima riga (#!/usr/bin/env python3) indica che lo script deve essere eseguito con l'interprete Python 3.
* Il commento # -\*- coding: utf-8 -\*- specifica che il file è codificato in UTF-8.
* La stringa di documentazione (""" ... """) descrive brevemente lo scopo dello script.
* Le importazioni includono moduli standard (os, pandas, argparse) e funzioni personalizzate (load\_sequence, get\_protein\_sequence, plot\_variant\_distribution, plot\_variant\_types, generate\_summary\_report) dai moduli sequence\_utils.py e visualize.py.
* **2. Definizione delle directory**



* DATA\_DIR e RESULTS\_DIR definiscono i percorsi delle directory per i dati di input e i risultati, rispettivamente:
  + DATA\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_))): Determina il percorso della directory principale del progetto (due livelli superiori alla directory in cui si trova questo script). Questo è un modo flessibile per fare riferimento a una directory di dati.
  + RESULTS\_DIR = os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_))), "results"): Definisce il percorso della directory "results" all'interno della directory principale del progetto, dove verranno salvati i risultati dell'analisi.
* **3. Parsing degli argomenti della riga di comando**

A computer code with text

AI-generated content may be incorrect.

def parse\_arguments(): Definisce una funzione per gestire gli argomenti da riga di comando.

parser = argparse.ArgumentParser(...): Crea un oggetto ArgumentParser con una descrizione dello script.

parser.add\_argument(...): Definisce gli argomenti che l'utente deve fornire:

* --gene: L'identificatore del gene da analizzare (obbligatorio).
* --sequence\_file: Il percorso del file FASTA contenente la sequenza del gene (obbligatorio).
* --variants\_file: Il percorso del file CSV contenente i dati delle varianti (obbligatorio).

return parser.parse\_args(): Analizza gli argomenti e restituisce un oggetto contenente i valori forniti.

* **4. Caricamento delle varianti**

A computer code with green and black text

AI-generated content may be incorrect.

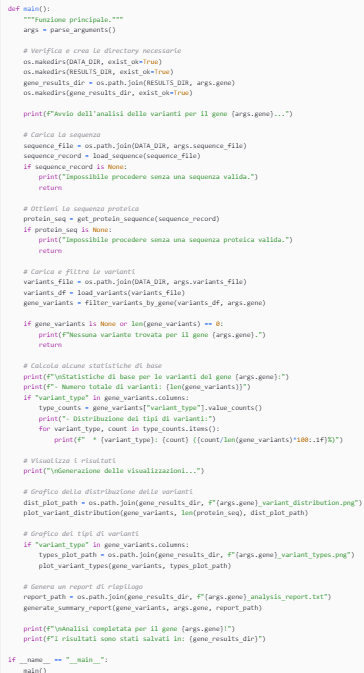
* La funzione load\_variants legge un file CSV contenente varianti genetiche e lo carica in un DataFrame di pandas. Se c'è un errore durante il caricamento, viene stampato un messaggio di errore e la funzione restituisce None.

**5. Filtro delle varianti per gene**

A screen shot of a computer code

AI-generated content may be incorrect.

* def filter\_variants\_by\_gene(variants\_df, gene\_id): Definisce una funzione per filtrare un DataFrame di varianti in base al nome del gene. Questa funzione è stata implementata per permettere di fare l’analisi su differenti geni simultaneamente.
* Verifica se il DataFrame in input è None e, in tal caso, restituisce None.
* Utilizza l'indicizzazione booleana di pandas (variants\_df["gene"] == gene\_id) per selezionare solo le righe in cui la colonna "gene" corrisponde all'gene\_id fornito. .copy() viene utilizzato per creare una nuova copia del DataFrame filtrato per evitare modifiche accidentali al DataFrame originale.
* Stampa un messaggio indicando il numero di varianti filtrate per il gene specificato.
* Restituisce il DataFrame contenente solo le varianti del gene specificato.
* **6. Funzione principale main**



def main(): Definisce la funzione principale che orchestra l'intero processo di analisi.

args = parse\_arguments(): Ottiene gli argomenti da riga di comando.

**Gestione delle directory:** Crea le directory DATA\_DIR, RESULTS\_DIR, e una sottodirectory specifica per il gene analizzato all'interno di RESULTS\_DIR, assicurandosi che esistano (exist\_ok=True).

Stampa un messaggio di avvio dell'analisi.

**Caricamento della sequenza:** Costruisce il percorso completo al file FASTA della sequenza e utilizza la funzione load\_sequence per caricarla. Gestisce il caso in cui la sequenza non possa essere caricata.

**Ottenimento della sequenza proteica:** Utilizza la funzione get\_protein\_sequence per ottenere la sequenza proteica dal record della sequenza caricata. Gestisce il caso in cui la sequenza proteica non possa essere ottenuta.

**Caricamento e filtraggio delle varianti:** Costruisce il percorso completo al file CSV delle varianti, carica le varianti utilizzando load\_variants, e poi filtra il DataFrame per il gene specificato utilizzando filter\_variants\_by\_gene. Gestisce il caso in cui non vengano trovate varianti per il gene.

**Calcolo di statistiche di base:** Calcola e stampa il numero totale di varianti per il gene e, se la colonna "variant\_type" è presente, calcola e stampa la distribuzione dei tipi di varianti.

**Generazione delle visualizzazioni:** Costruisce i percorsi per i file dei grafici (distribuzione delle varianti e tipi di varianti) all'interno della directory dei risultati del gene e chiama le funzioni plot\_variant\_distribution e plot\_variant\_types per generare e salvare i grafici. La generazione del grafico dei tipi è condizionale alla presenza della colonna "variant\_type".

**Generazione del report di riepilogo:** Costruisce il percorso per il file del report e chiama la funzione generate\_summary\_report per creare e salvare un report di testo con un riepilogo dell'analisi.

Stampa un messaggio di completamento dell'analisi e indica dove sono stati salvati i risultati.

* **7. Entry point dello script**

A group of colorful lines

AI-generated content may be incorrect.

* Questa riga assicura che la funzione main venga eseguita solo se lo script viene eseguito direttamente, e non se viene importato come modulo in un altro script.

**sequence\_utils.py**

Lo script fornisce due funzioni principali: una per caricare sequenze da file in vari formati e una per cercare di ottenere la sequenza proteica corrispondente, gestendo anche il caso in cui la sequenza in input sia DNA e necessiti di una traduzione (semplificata).

* **8. Intestazione dello script**

**Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.**

* #!/usr/bin/env python3: Questa è una "shebang line". Indica al sistema operativo che questo script deve essere eseguito con l'interprete Python 3. È particolarmente utile su sistemi Unix-like per rendere lo script eseguibile direttamente.
* # -\*- coding: utf-8 -\*-: Questa linea specifica la codifica dei caratteri sorgente come UTF-8. Ciò permette di includere caratteri Unicode (come quelli accentati) all'interno del codice.
* """Modulo per la gestione delle sequenze genetiche.""": Questa è una docstring (document string) a livello di modulo. Fornisce una breve descrizione dello scopo generale del modulo. Gli strumenti di documentazione possono utilizzare questa stringa per generare documentazione automatica.
* **9. Intestazione dello script**

Immagine che contiene Carattere, Elementi grafici, testo, tipografia

Il contenuto generato dall'IA potrebbe non essere corretto.

* from Bio import SeqIO: Questa riga importa il modulo SeqIO dalla libreria Biopython. SeqIO è un modulo fondamentale per leggere e scrivere file di sequenze biologiche in vari formati (come FASTA, GenBank, ecc.). Fornisce funzionalità per iterare attraverso le sequenze in un file e per leggere una singola sequenza.
* import os: Questa riga importa il modulo os della libreria standard di Python. Il modulo os fornisce un modo per interagire con il sistema operativo, ad esempio per manipolare i percorsi dei file.
* **10. Definizione della funzione load\_sequence**

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

* def load\_sequence(file\_path, format="fasta"): Questa linea definisce una funzione chiamata load\_sequence che accetta due argomenti:
* file\_path: Una stringa che rappresenta il percorso al file contenente la sequenza.
* format: Una stringa opzionale che specifica il formato del file. Se non fornito, il valore predefinito è "fasta".
* Try:
  + record = SeqIO.read(file\_path, format): Questa è la linea chiave. Utilizza la funzione SeqIO.read() di Biopython per leggere una singola sequenza dal file specificato nel file\_path nel formato indicato da format. Il risultato è un oggetto SeqRecord, che contiene la sequenza stessa (come un oggetto Seq) e metadati associati (come l'ID, la descrizione, ecc.).
  + print(f"Sequenza caricata: {record.id}, lunghezza: {len(record.seq)} bp"): Questa linea stampa un messaggio informativo che indica che la sequenza è stata caricata con successo, mostrando l'ID della sequenza e la sua lunghezza in basi (bp).
  + return record: Se la lettura del file ha successo, la funzione restituisce l'oggetto SeqRecord contenente la sequenza.
* except Exception as e: Questo blocco gestisce le potenziali eccezioni (errori) che potrebbero verificarsi durante il tentativo di leggere il file. Se si verifica un errore (ad esempio, se il file non esiste o se il formato specificato non è corretto), il codice all'interno del blocco except viene eseguito.
  + print(f"Errore durante il caricamento della sequenza: {str(e)}"): Se si verifica un'eccezione, questa linea stampa un messaggio di errore che include una descrizione dell'errore (str(e)).
  + return None: In caso di errore, la funzione restituisce None per indicare che la sequenza non è stata caricata correttamente.
* **11. Definizione della funzione get\_protein\_sequence**

Immagine che contiene testo, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.

* def get\_protein\_sequence(sequence\_record): Questa linea definisce una funzione chiamata get\_protein\_sequence che accetta un argomento:
* sequence\_record: Un oggetto SeqRecord (come quello restituito da load\_sequence).
* seq = sequence\_record.seq: Questa linea estrae l'oggetto Seq (che rappresenta la sequenza biologica) dall'oggetto SeqRecord e lo assegna alla variabile seq.
* if all(c in 'ATGCNatgcn' for c in seq): Questa è una condizione che verifica se tutti i caratteri nella sequenza (seq) sono presenti nell'insieme dei nucleotidi del DNA (sia maiuscoli che minuscoli). Se questa condizione è vera, si assume che la sequenza sia DNA e si tenta la traduzione in proteina.
* for i in range(3): Questo ciclo for itera attraverso i primi tre possibili frame di lettura (0, 1 e 2) di una sequenza di DNA.
  + prot = seq[i:].translate(to\_stop=True): Per ciascun frame di lettura, questa linea esegue la traduzione della sequenza di DNA in una sequenza proteica utilizzando il metodo translate() dell'oggetto Seq. L'argomento to\_stop=True fa sì che la traduzione si interrompa al primo codone di stop incontrato.
  + if len(prot) > 100: Questa condizione verifica se la sequenza proteica tradotta (prot) ha una lunghezza maggiore di 100 aminoacidi. Questa è un'euristica semplice per filtrare potenziali frammenti non significativi o traduzioni parziali dato che le proteine funzionali tendono ad avere una determinata lunghezza minima.
  + return prot: Se viene trovata una sequenza proteica tradotta che soddisfa la condizione di lunghezza, la funzione la restituisce come una stringa.
  + print("Nessuna proteina valida trovata nella traduzione"): Se il ciclo for completa tutte e tre le iterazioni senza trovare una proteina tradotta di lunghezza sufficiente, viene stampato questo messaggio.
  + return None: In questo caso, la funzione restituisce None per indicare che non è stata trovata una sequenza proteica valida dalla traduzione.
* else: Se la condizione iniziale (if all(c in 'ATGCNatgcn' for c in seq):) è falsa, si assume che la sequenza fornita sia già una sequenza proteica.
* return seq: In questo caso, la funzione restituisce direttamente la sequenza (seq) senza effettuare alcuna traduzione.

**visualize.py**

Questo script fornisce tre funzioni principali per analizzare e visualizzare dati di varianti genetiche: una per visualizzare la distribuzione delle varianti lungo una proteina, una per mostrare la proporzione dei diversi tipi di varianti e una per generare un report di testo riassuntivo con statistiche chiave.

* **12. Intestazione dello script**

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

* Le prime due righe sono le stesse del modulo precedente, indicando l'interprete Python e la codifica dei caratteri.
* La docstring descrive lo scopo di questo modulo: la visualizzazione delle varianti genetiche.
* **13. Importazione di moduli**

Immagine che contiene Carattere, testo, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.

* import matplotlib.pyplot as plt: Importa il sottomodulo pyplot della libreria matplotlib, utilizzata per la creazione di grafici (istogrammi e grafici a torta in questo caso). Viene convenzionalmente importato con l'alias plt.
* import pandas as pd: Importa la libreria pandas, fondamentale per la manipolazione e l'analisi di dati tabellari, rappresentati qui come DataFrame. Viene convenzionalmente importata con l'alias pd.
* import os: Anche in questo caso, importa il modulo os per interagire con il sistema operativo.
* **13. Definizione della funzione plot\_variant\_distribution**

Immagine che contiene testo, schermata, software

Il contenuto generato dall'IA potrebbe non essere corretto.

* def plot\_variant\_distribution(variants\_df, protein\_length, output\_path): Definisce una funzione che prende in input un DataFrame di varianti, la lunghezza della proteina e il percorso dove salvare il grafico.
* positions = variants\_df["position"].dropna(): Estrae la colonna "position" dal DataFrame e utilizza .dropna() per rimuovere eventuali valori mancanti (NaN) che potrebbero essere presenti.
* plt.figure(figsize=(12, 6)): Crea una nuova figura per il grafico con una dimensione specificata (larghezza 12 pollici, altezza 6 pollici).
* plt.hist(...): Genera un istogramma.
  + positions: I dati da visualizzare (le posizioni delle varianti).
  + bins=range(0, protein\_length+10, 10): Definisce gli intervalli (bin) dell'istogramma. Crea una sequenza di numeri da 0 alla lunghezza della proteina più 10, con un passo di 10. Questo raggruppa le varianti in intervalli di 10 aminoacidi lungo la proteina.
  + edgecolor='black': Imposta il colore del bordo delle barre dell'istogramma a nero.
  + alpha=0.7: Imposta la trasparenza delle barre a 0.7.
* plt.title(...), plt.xlabel(...), plt.ylabel(...): Impostano il titolo del grafico e le etichette degli assi x e y.
* plt.grid(True, linestyle="--", alpha=0.7): Aggiunge una griglia al grafico per facilitare la lettura dei valori.
* plt.tight\_layout(): Regola automaticamente il layout dei subplot per fornire uno spazio sufficiente.
* plt.savefig(...): Salva il grafico su disco nel percorso specificato (output\_path).
  + dpi=300: Imposta la risoluzione dell'immagine a 300 punti per pollice (alta qualità).
  + bbox\_inches="tight": Elimina lo spazio bianco extra attorno al grafico.
* plt.close(): Chiude la figura per liberare memoria.
* Il blocco try...except gestisce eventuali errori durante la creazione del grafico.
* **14. Definizione della funzione plot\_variant\_types**

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

* def plot\_variant\_types(variants\_df, output\_path): Definisce una funzione per creare un grafico a torta che mostra la proporzione dei diversi tipi di varianti.
* type\_counts = variants\_df["variant\_type"].value\_counts(): Utilizza il metodo .value\_counts() di pandas per contare la frequenza di ciascun valore unico nella colonna "variant\_type" del DataFrame.
* plt.pie(...): Genera un grafico a torta.
  + type\_counts: Le frequenze dei diversi tipi di varianti.
  + labels=type\_counts.index: Le etichette per ciascuna fetta del grafico (i nomi dei tipi di varianti).
  + autopct="%1.1f%%": Formatta la percentuale visualizzata su ciascuna fetta con una cifra decimale.
  + startangle=90: Inizia la prima fetta a 90 gradi (in alto).
  + shadow=True: Aggiunge un'ombra al grafico per un effetto visivo.
  + labeldistance=1.15: Imposta la distanza delle etichette dal centro del grafico.
  + pctdistance=0.85: Imposta la distanza delle percentuali dal centro del grafico.
* plt.title(...): Imposta il titolo del grafico.
* plt.tight\_layout(): Regola il layout.
* plt.savefig(...) e plt.close(): Salvano e chiudono la figura, in modo simile alla funzione precedente.
* print(f"Grafico dei tipi di varianti salvato in: {output\_path}") Stampa un messaggio di conferma.
* **15. Definizione della funzione generate\_summary\_report**

Immagine che contiene testo, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.

* def generate\_summary\_report(variants\_df, gene\_id, output\_path): Definisce una funzione per creare un report di testo riassuntivo. Prende in input il DataFrame delle varianti, l'ID del gene e il percorso del file di output.
* with open(output\_path, "w") as f: Apre un file nel percorso specificato in modalità scrittura ("w"). L'uso di with assicura che il file venga chiuso correttamente anche in caso di errori. La variabile f rappresenta l'oggetto file.
* f.write(...): Scrive diverse righe di testo nel file, formattando le informazioni rilevanti:
  + Titolo del report con l'ID del gene.
  + Numero totale di varianti.
  + Distribuzione dei tipi di varianti (conteggio e percentuale).
  + Se la colonna "pathogenicity" è presente, riassume anche la distribuzione della patogenicità.
  + Le 5 posizioni con il maggior numero di varianti.

variants\_df["variant\_type"].value\_counts(): Come nella funzione precedente, conta i tipi di varianti.

variants\_df["pathogenicity"].value\_counts(): Se la colonna esiste, conta le occorrenze di ciascun valore di patogenicità.

variants\_df["position"].value\_counts().head(5): Conta le varianti per ciascuna posizione e .head(5) seleziona le 5 posizioni con il conteggio più alto.

print (f"Report di riepilogo salvato in: {output\_path}") Stampa un messaggio di conferma che indica dove è stato salvato il report.

**generate\_scn5a\_variants**

Questo script è uno strumento per generare dati di varianti genetiche simulate. Utilizza la libreria argparse per gestire gli input da riga di comando, Biopython per caricare sequenze FASTA e pandas per organizzare e salvare i dati in formato CSV. La logica di generazione delle varianti è basata su probabilità definite per diversi tipi di varianti e categorie di patogenicità, rendendo i dati simulati statisticamente plausibili.

* **16. Intestazione dello script**

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Come negli script precedenti, queste righe specificano l'interprete Python e la codifica dei caratteri.

La docstring descrive lo scopo di questo script: generare dati di esempio per le varianti di SCN5A.

* **17. Importazione di moduli**

Immagine che contiene testo, Carattere, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.

* import random: Importa il modulo random per generare numeri casuali, utile per simulare le posizioni e i tipi di varianti.
* import pandas as pd: Importa la libreria pandas per creare e manipolare DataFrame, che verranno utilizzati per organizzare i dati delle varianti e salvarli in formato CSV.
* import os: Importa il modulo os, sebbene non sia direttamente utilizzato nelle funzioni fornite. Potrebbe essere previsto per operazioni future relative al sistema operativo.
* import argparse: Importa il modulo argparse, che facilita la definizione e la gestione degli argomenti da riga di comando.
* from Bio import SeqIO: Importa il modulo SeqIO da Biopython per leggere file di sequenze (in questo caso, file FASTA).
* from Bio.Seq import Seq: Importa la classe Seq da Biopython.
* **17. Definizione della funzione parse\_arguments**

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

def parse\_arguments(): Definisce una funzione per gestire gli argomenti forniti all'esecuzione dello script da riga di comando.

parser = argparse.ArgumentParser(...): Crea un oggetto ArgumentParser con una descrizione dello script.

parser.add\_argument(...): Definisce i diversi argomenti che l'utente può specificare:

* --fasta: Percorso del file FASTA contenente la sequenza di SCN5A (obbligatorio, required=True).
* --output: Nome del file CSV di output (opzionale, predefinito a "variants.csv").
* --num\_variants: Numero di varianti da generare (opzionale, predefinito a 20).
* --random\_seed: Seed per il generatore di numeri casuali (opzionale, predefinito a 42). L'uso di un seed fisso rende la generazione riproducibile.

return parser.parse\_args(): Analizza gli argomenti della riga di comando e restituisce un oggetto contenente i valori forniti dall'utente (o i valori predefiniti).

* **18. Definizione della funzione load\_sequence**

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

def load\_sequence(fasta\_file): Definisce una funzione che prende in input il percorso di un file FASTA.

record = SeqIO.read(fasta\_file, "fasta"): Utilizza SeqIO.read() per leggere la sequenza dal file FASTA. Si assume che il file contenga una singola sequenza.

protein\_seq = record.seq.translate(): Traduce la sequenza nucleotidica (record.seq) in una sequenza proteica utilizzando il metodo .translate() dell'oggetto Seq. Si assume che la sequenza FASTA fornita sia la sequenza codificante (CDS). La traduzione si fermerà a un codone di stop.

Il blocco try...except gestisce eventuali errori durante il caricamento del file.

Restituisce la sequenza proteica tradotta o None in caso di errore.

* **19. Definizione della funzione generate\_variants**

Immagine che contiene testo, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.

def generate\_variants(protein\_seq, num\_variants, random\_seed=42): Definisce la funzione che genera le varianti simulate. Prende la sequenza proteica, il numero di varianti da generare e un seed opzionale per la randomizzazione.

random.seed(random\_seed): Inizializza il generatore di numeri casuali con il seed fornito, garantendo la riproducibilità.

variant\_types e pathogenicity\_types: Dizionari che definiscono i possibili tipi di varianti e le categorie di patogenicità, insieme alle loro probabilità di occorrenza.

amino\_acids: Una stringa contenente gli aminoacidi standard.

Il ciclo for \_ in range(num\_variants): Itera per il numero specificato di varianti da generare.

All'interno del ciclo:

* position = random.randint(1, protein\_length): Sceglie una posizione casuale lungo la proteina.
* original\_aa = protein\_seq[position-1]: Ottiene l'aminoacido originale nella posizione scelta (si noti l'indice -1 perché le stringhe in Python sono a base 0, mentre le posizioni aminoacidiche sono generalmente a base 1).
* variant\_type = random.choices(...): Sceglie un tipo di variante casuale basato sulle probabilità definite in variant\_types.
* pathogenicity = random.choices(...): Sceglie una categoria di patogenicità casuale basata sulle probabilità in pathogenicity\_types.
* La serie di istruzioni if/elif determina l'aminoacido o la descrizione della variante (variant\_aa) in base al variant\_type:
  + missense: Sceglie un nuovo aminoacido diverso da quello originale.
  + nonsense: Imposta la variante come un codone di stop ("X").
  + frameshift: Imposta la variante come "fs".
  + inframe\_deletion: Simula una delezione di 1-3 aminoacidi.
* Un dizionario contenente le informazioni della variante viene creato e aggiunto alla lista variants.

La funzione restituisce la lista di dizionari variants.

* **20. Definizione della funzione main**

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

def main(): Definisce la funzione principale che viene eseguita quando lo script è avviato.

args = parse\_arguments(): Chiama la funzione per analizzare gli argomenti da riga di comando.

protein\_seq = load\_sequence(args.fasta): Carica la sequenza proteica dal file FASTA specificato.

Un controllo if protein\_seq is None: verifica se la sequenza è stata caricata correttamente. In caso contrario, stampa un messaggio di errore ed esce dalla funzione.

Stampa un messaggio indicando la lunghezza della sequenza proteica caricata.

variants = generate\_variants(...): Chiama la funzione per generare le varianti utilizzando la sequenza proteica e i parametri forniti.

variants\_df = pd.DataFrame(variants): Crea un DataFrame pandas dalla lista di dizionari variants. Le chiavi dei dizionari diventano le intestazioni delle colonne del DataFrame.

variants\_df.to\_csv(args.output, index=False): Salva il DataFrame in un file CSV nel percorso specificato (args.output). index=False impedisce la scrittura dell'indice del DataFrame nel file CSV.

Stampa un messaggio di conferma indicando il numero di varianti generate e il nome del file CSV di output.

if \_\_name\_\_ == "\_\_main\_\_": main(): Assicura che la funzione main() venga chiamata solo quando lo script viene eseguito direttamente (non quando viene importato come un modulo).