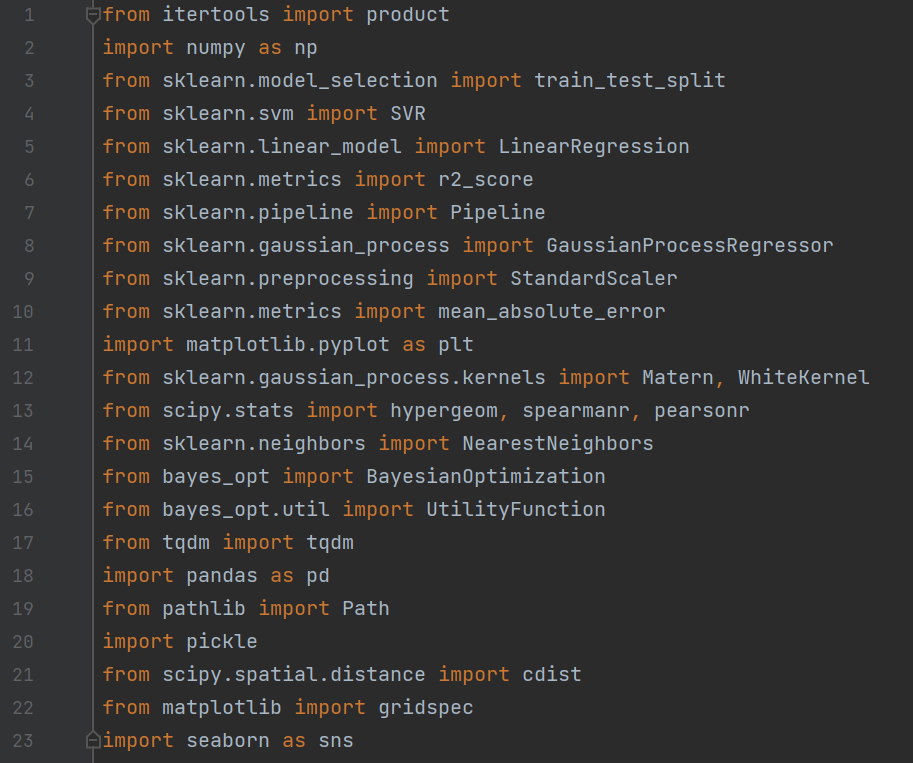
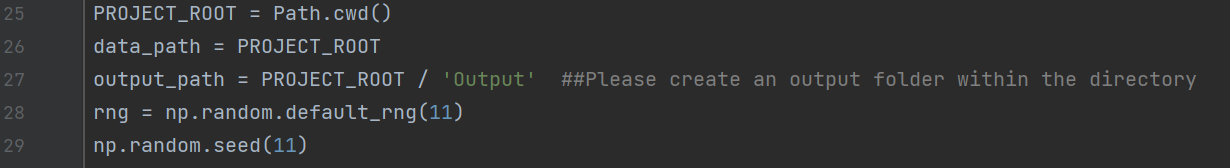
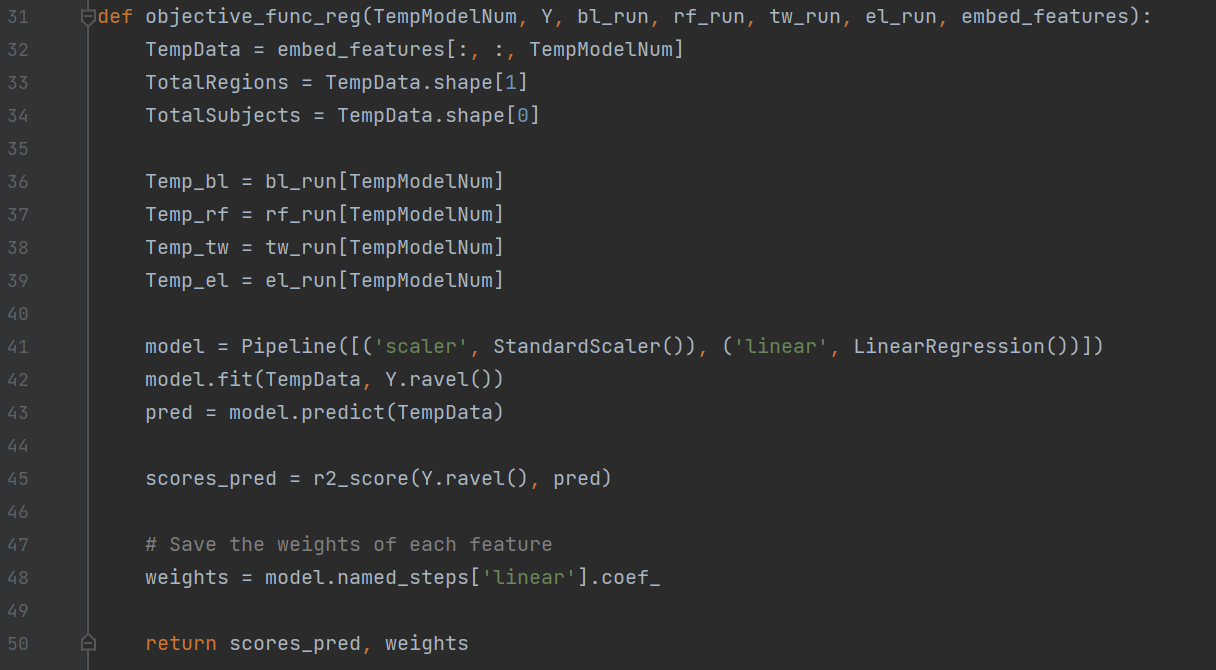
**Supplementary 2: The Structure and Functionality of the MultiverseSampling\_Helper.py Script**

**Preliminary Installations**

1. sklearn is used for machine learning tasks, including data splitting, egression models, and evaluation metrics; the kernel definitions for Gaussian Process regression, and for statistical tests and correlation metrics;
2. Sklearn.neighbors and scipy.spatial are used for identifying the nearest points in the model embedding space;
3. bayesopt libraries implements the core Bayesian Optimization process functions;
4. numpy, pandas and pickle libraries assist in organizing, reading and manipulating data;
5. pathlib manages file paths;
6. matplotlib and seaborn libraries are used to create visual plots and graphs;
7. tqdm creates progress bars for loops;
8. itertools generates the Cartesian product of multiple input sequences for grid construction.

**Preparation**

1. PROJECT\_ROOT defines the root directory for the project;
2. data\_path and output\_path are set for organizing input data and outputs;
3. The random seed ensures reproducibility in random number generation.

**Function objective\_func\_reg**

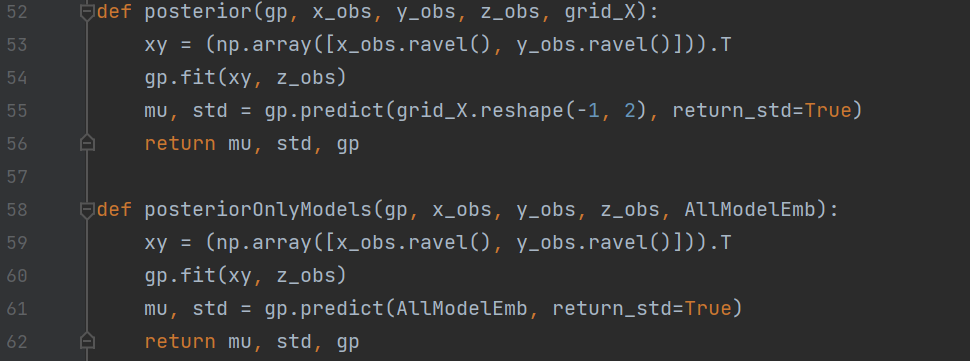
This function evaluates the performance of a pipeline model by fitting a linear regression to the feature data and computing the *R*2.

Inputs:

1. TempModelNum – index of the model to evaluate;
2. Y – target values;
3. bl\_run, rf\_run, tw\_run, el\_run – parameters corresponding to decision nodes;
4. embed\_features – feature embedding data for models.

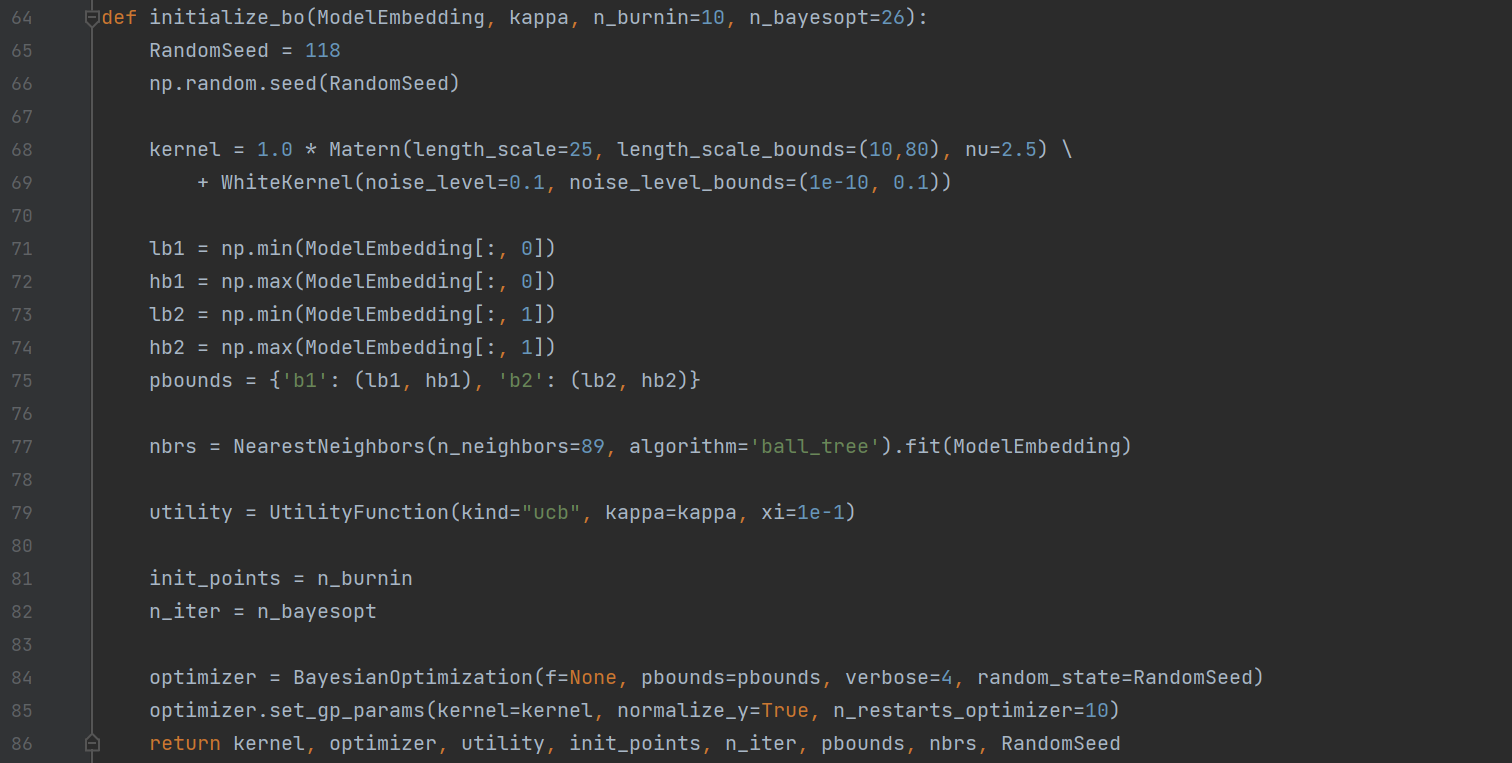
Steps:

1. Extracts features for a specific model;
2. Constructs and trains a linear regression model;
3. Computes the *R*2 to evaluate the model’s performance;
4. Returns the *R*2 and feature weights.

**Function posterior**

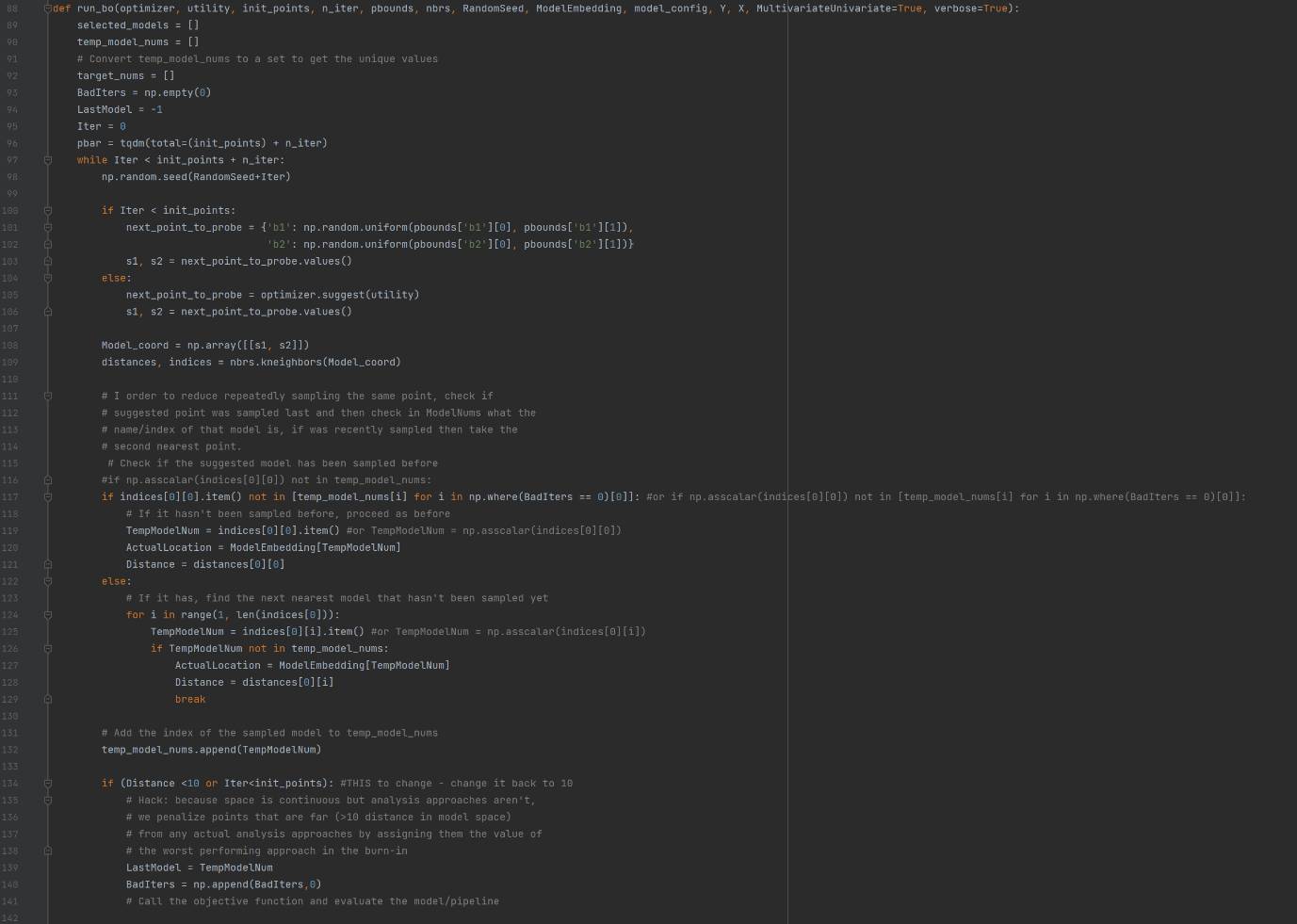
Computes the posterior mean and standard deviation for Gaussian Process regression.

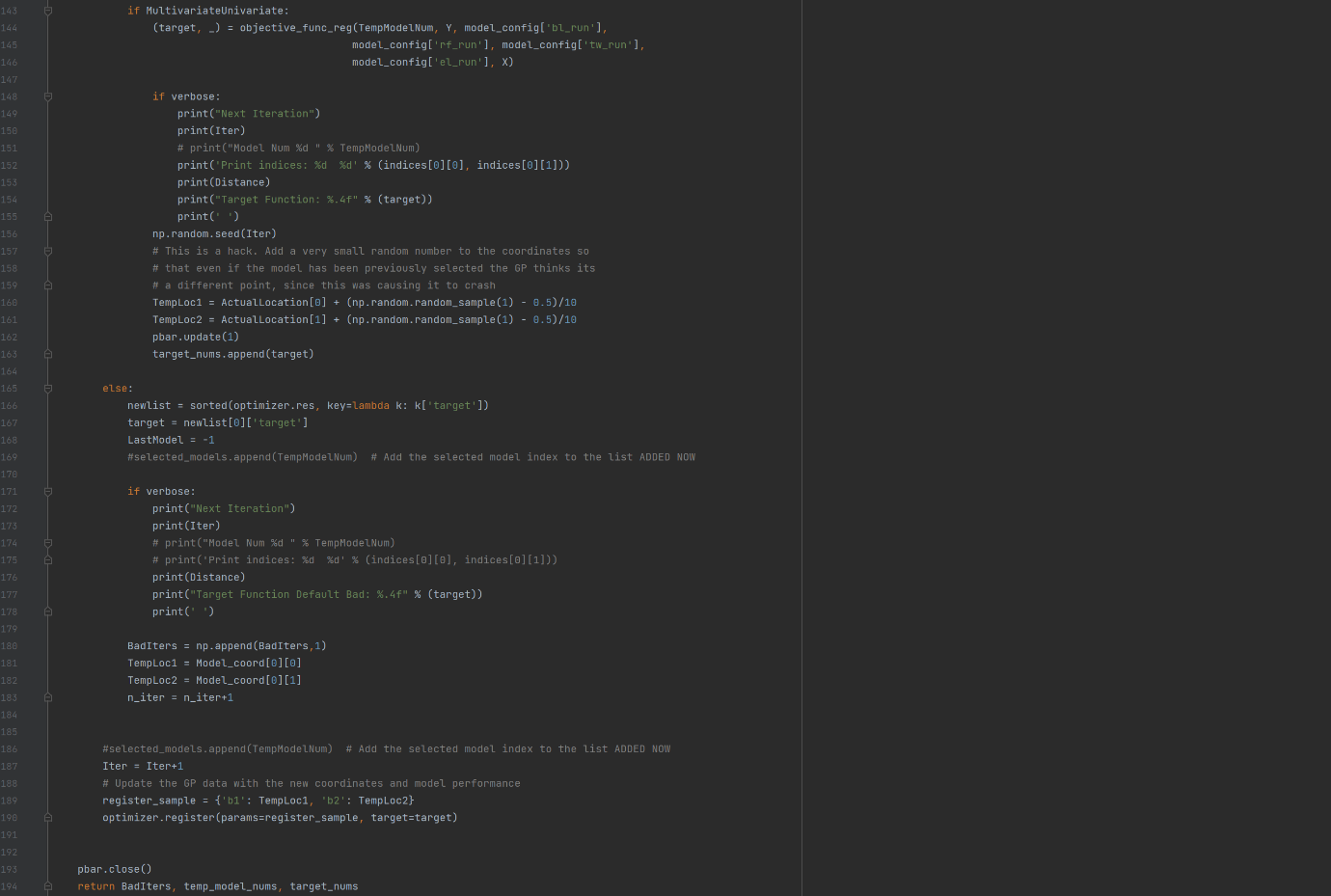
Fits the Gaussian Process to observed data points (x\_obs, y\_obs, z\_obs), which dynamically expands as more points are sampled, and predicts the posterior distribution for new points (grid\_x).

**Function initialize\_bo**

This function sets up the Bayesian optimization:

1. Defines a kernel combining a Matern kernel (spatial covariance) and noise;
2. Configures bounds for optimization and initializes the optimizer and utility function;
3. Uses NearestNeighbors for model selection based on proximity.

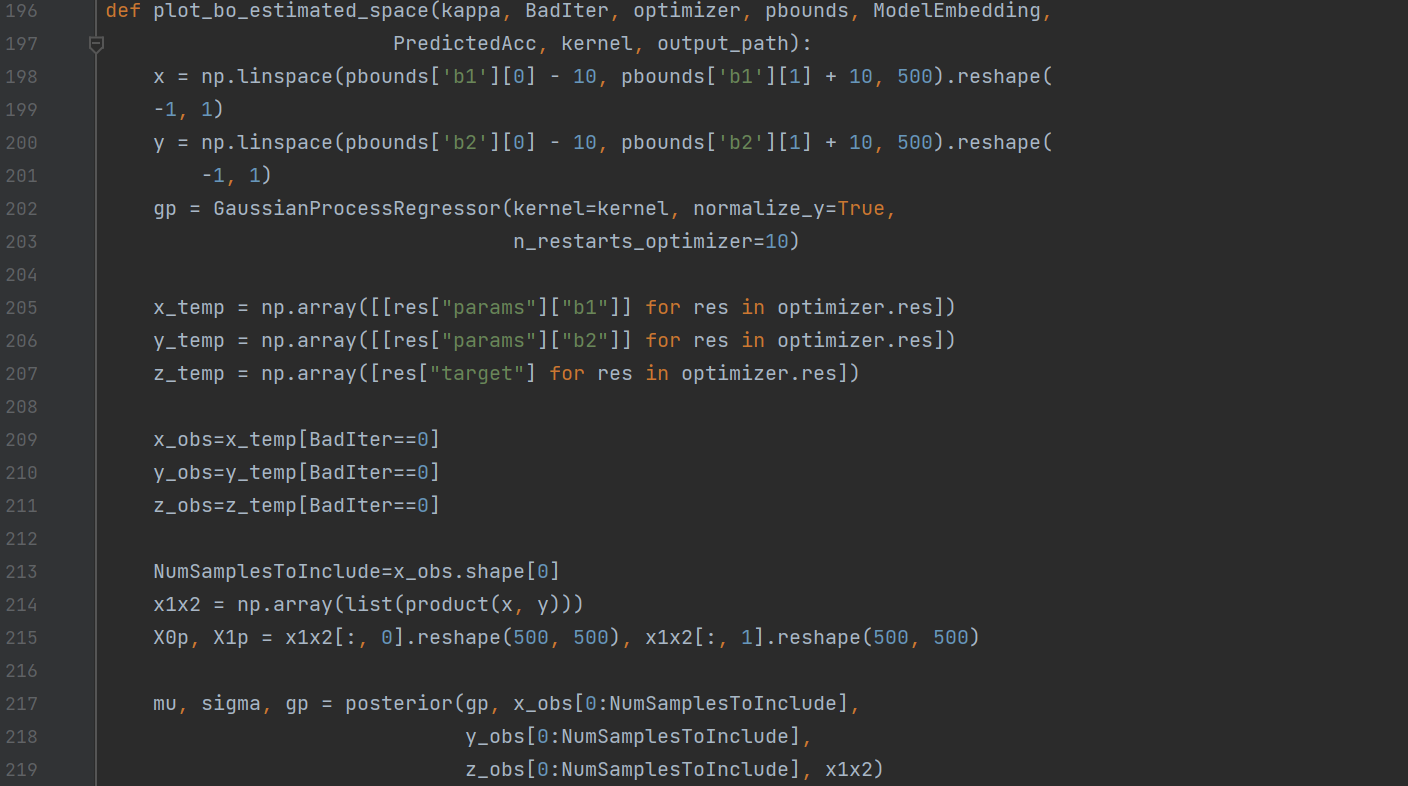
**Function run\_bo**

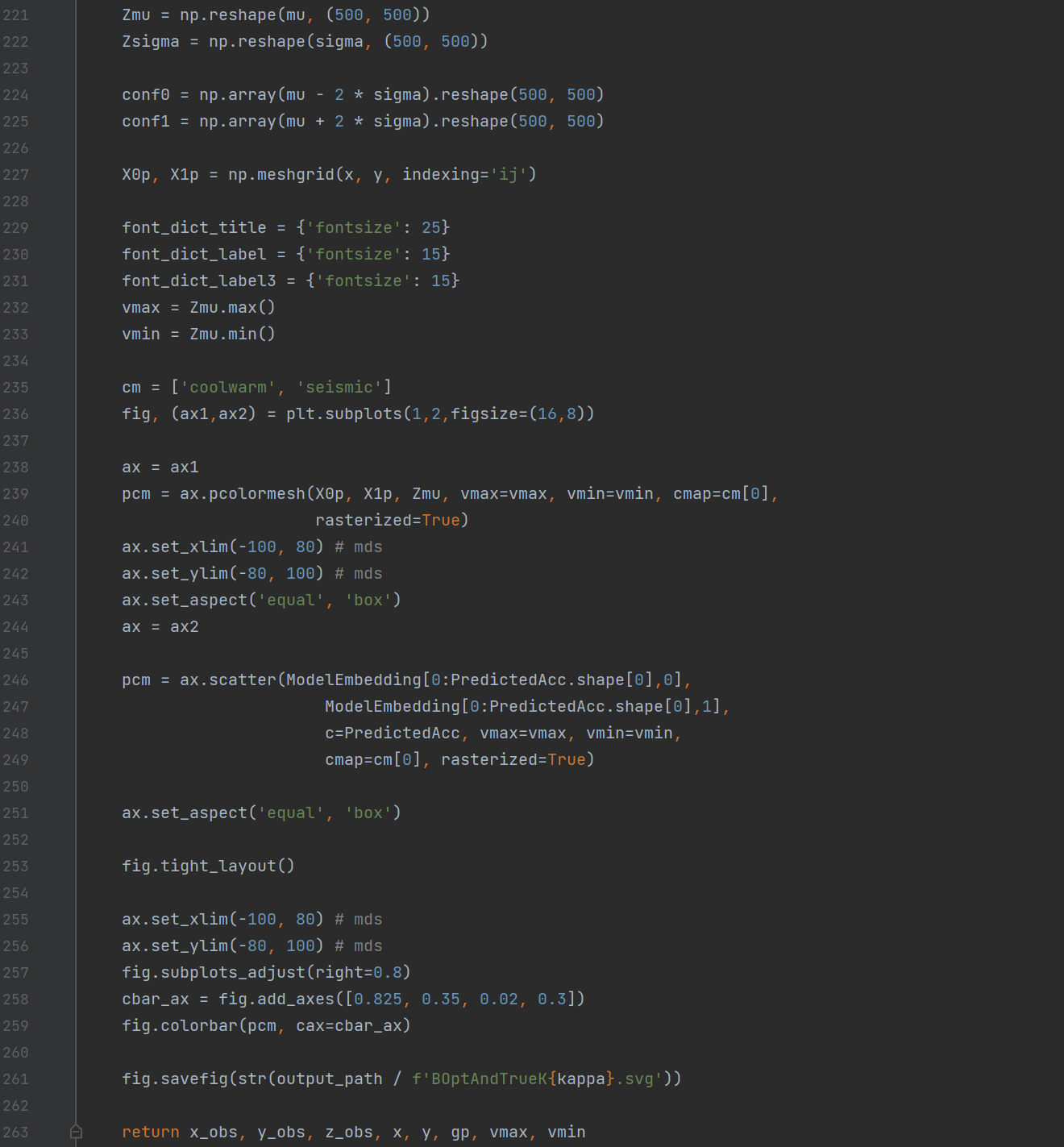


Implements Bayesian Optimization for active learning:

1. Randomly samples points (init\_points) during a "burn-in" phase;
2. Suggests new points using the utility function;
3. Evaluates the objective function for the sampled models and updates the optimizer;
4. Penalizes models that were recently sampled or far from existing points to avoid repeated sampling and maintain exploration;
5. Tracks bad iterations, selected models, and target scores;
6. The use of np.random.seed(RandomSeed + Iter) ensures reproducibility for each iteration.

**Function plot\_bo\_estimated\_space**



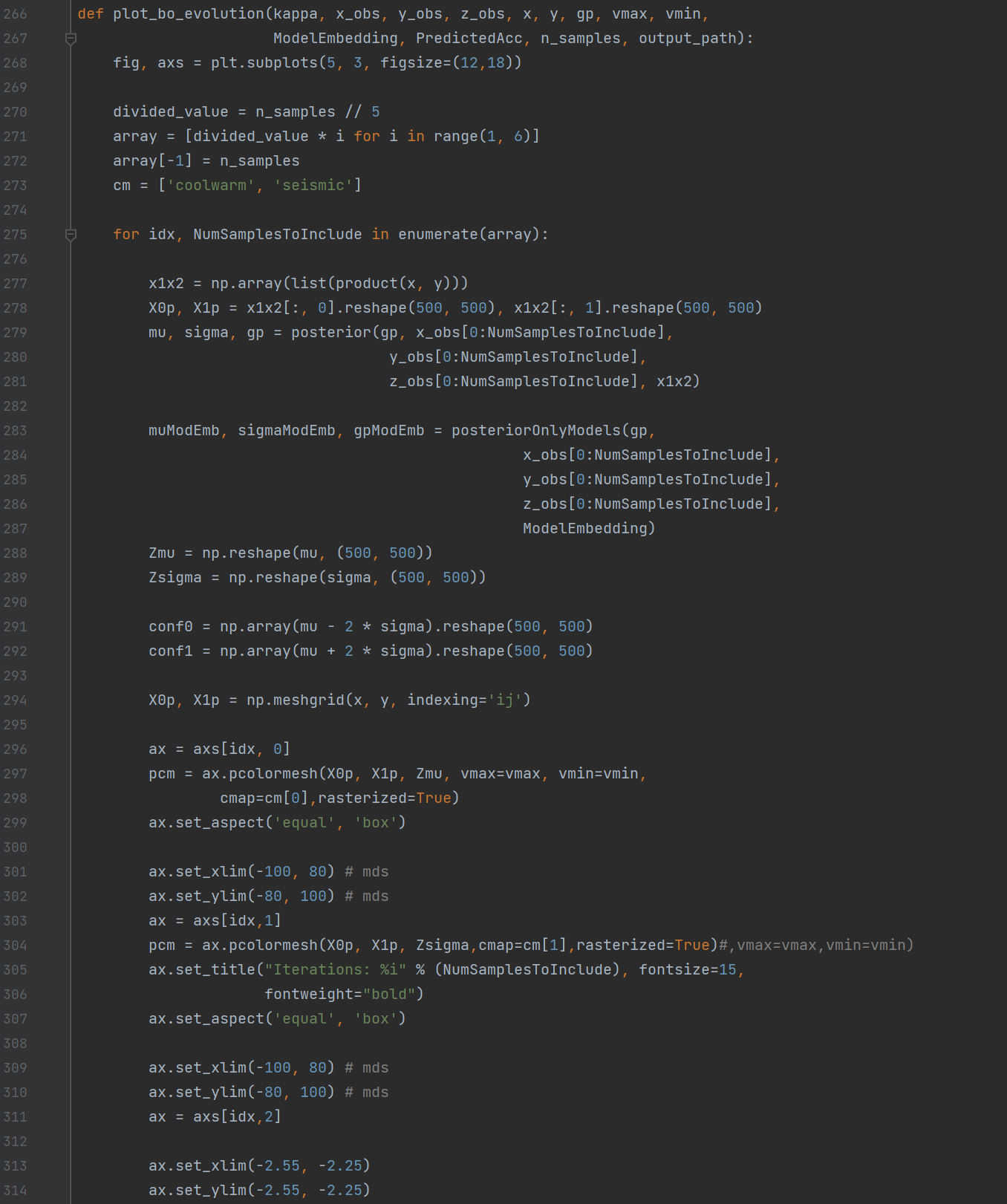


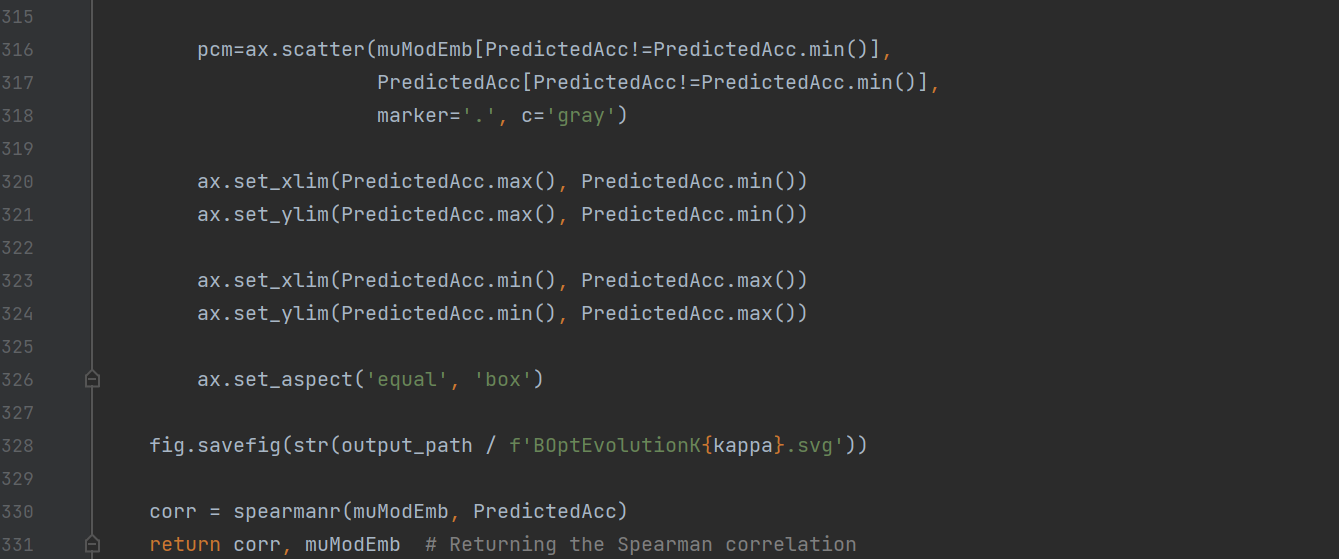
Visualizes the Bayesian Optimization process:

1. Plots the posterior mean and uncertainty of the search space;
2. Highlights the locations of sampled models.

Specifically,

* The np.linspace funtion creates evenly spaced grid points in the parameter space, defined by pbounds (the boundaries for b1 and b2). The bounds are expanded slightly (-10, +10) to provide better visualization at the edges;
* The GuassianProcessRegressor() initialized the Gaussian Process using the provided kernel (which includes spatial covariance and noise). The Gaussian Process will fit the observed data and predict the posterior mean and uncertainty across the grid;
* The x/y/z\_temp and x/y/z\_obs extract observed points (b1, b2) and their corresponding target values from the Bayesian optimizer’s history. BadIter == 0 excludes points flagged as bad iterations as defined above;
* x1x2 and X0p, X1p construct all combinations of grid points (x,y) using a Cartesian product, resulting in a 2D grid of shape (500,500);
* mu, sigma, gp calls the posterior function to fit the Gaussian Process to observed data, predicts the posterior mean and uncertainty over the grid, and reshapes predictions (Zmu, Zsigma) into 2D arrays for plotting;
* conf0 and conf1 compute 95% confidence intervals using +/- 2 SDs from the mean;
* Then plot parameters like font size, color map and value range are configured;
* Then the posterior mean and the predictive accuracies are plotted.

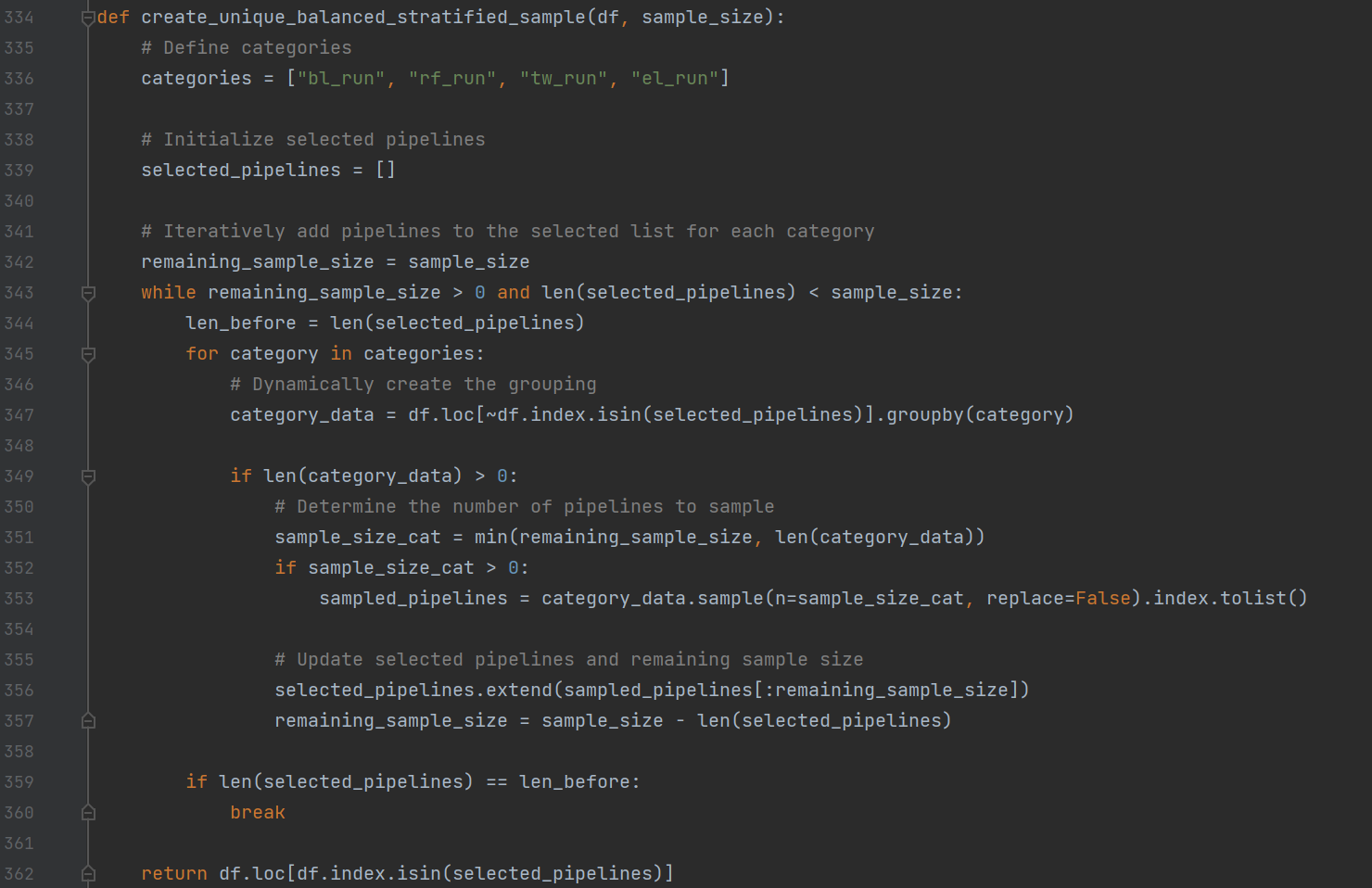
**Function plot\_bo\_evolution**

****

This function visualizes the evolution of the Bayesian optimization across iterations by:

1. Showing the Gaussian Process posterior predictions (mean and uncertainty) change as more observations are added.
   1. Creates a grid of 5 rows and 3 columns for plotting (3 columns are for the posterior mean, the posterior uncertainty, and the scatterplot for the predicted vs actual performance of the models;
   2. Divides the total number of samples (n\_samples) into 5 equal segments, to represent snapshots of the optimization process at different stages, and loops through these defined intervals to visualize the optimization at each stage;
   3. Creates a grid of points over the search space using the Cartesian product (x, y), which are used for the Gaussian Process predictions;
   4. Fits the Gaussian Process on the observed data (x\_obs, y\_obs, z\_obs) up to the current iteration, and predicts the posterior mean and uncertainty over the grid;
   5. Fits a Gaussian Process for the model embeddings (ModelEmbedding), and predicts the mean (muModEmb) and uncertainty (sigmaModEmb) for these embeddings;
   6. Plots the posterior mean (Zmu) and the posterior uncertainty (Zsigma) as heatmaps over the search space;
   7. Plots the predicted means (muModEmb) against the accuracies (PredictedAcc) for the model embeddings to compare the Gaussian Processes’ predictions with the actual performance of sampled models, and saves as an SVG file.
2. Calculating the Spearman correlation between predicted performance and actual performance of models.
   1. Computes the Spearman correlation between the Gaussian Process’ predicted means (muModEmb) and the actual accuracies (PredictedAcc);
   2. Returns the correlation and predicted means for further analysis.

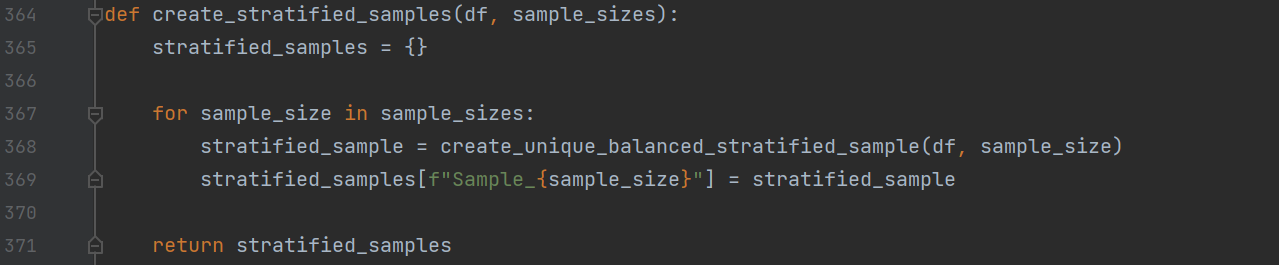
In summary, this function tracks the Bayesian optimization progress through visualization, provides three plots per stage (the posterior mean, the posterior uncertainty, and the predicted vs. actual performance), saves the evolution plot which visualizes the optimization process, and evaluates the Gaussian Process accuracy by returning the Spearman correlation between the predictions and actual performance.

**Function create\_unique\_balanced\_stratified\_sample**

This function generates a stratified sample that balances representation of options within each decision node.

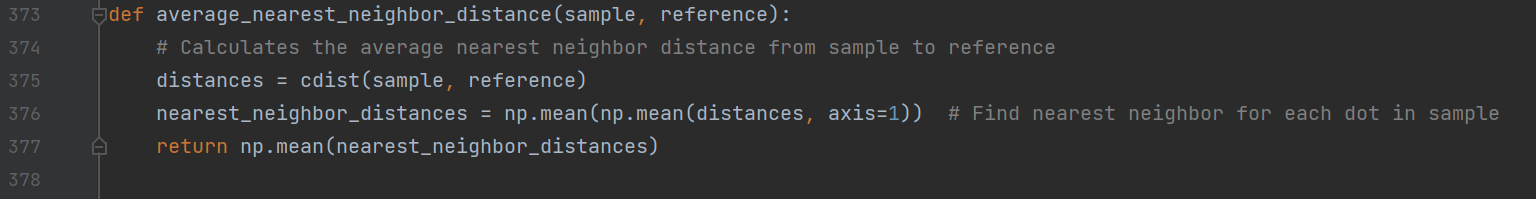
1. Defines categories (columns in the dataframe, which are the decision node categories);
2. Initializes the selected\_pipelines variable, which keeps track of the indices of rows that have already been selected, and the remaining\_sample\_size variable, which tracks how many more samples are needed to reach the target sample\_size;
3. Engages in a while loop until the target sample\_size is reached, by iterating until remaining\_sample\_size becomes 0, or the selected\_pipelines list reaches the desired sample\_size;
4. Groups the remaining data by each category by filtering the dataframe to exclude rows that are already selected and grouping the remaining rows by category;
5. Samples from each group by determining how many rows to sample within each group and randomly sampling within that group without replacement;
6. Updates the selected pipelines by adding the sampled rows to the selected\_pipelines list and updating the remaining\_sample\_size based on the number of rows added;
7. Considering an exit condition, which exits the loop early if no new rows are added during the current iteration to prevent an infinite loop in cases where there is insufficient data;
8. Returns the stratified sample containing only the rows corresponding to the selected pipelines.

If you do not wish for a balanced representation of options within a specific decision node, for example if this is not representative of your multiverse because there is a bias within the larger multiverse of defensible pipelines, the function should be amended accordingly.

**Function create\_stratified\_samples**

This function uses the create\_unique\_balanced\_stratified\_sample to generate multiple stratified samples of specific sizes from the dataset.

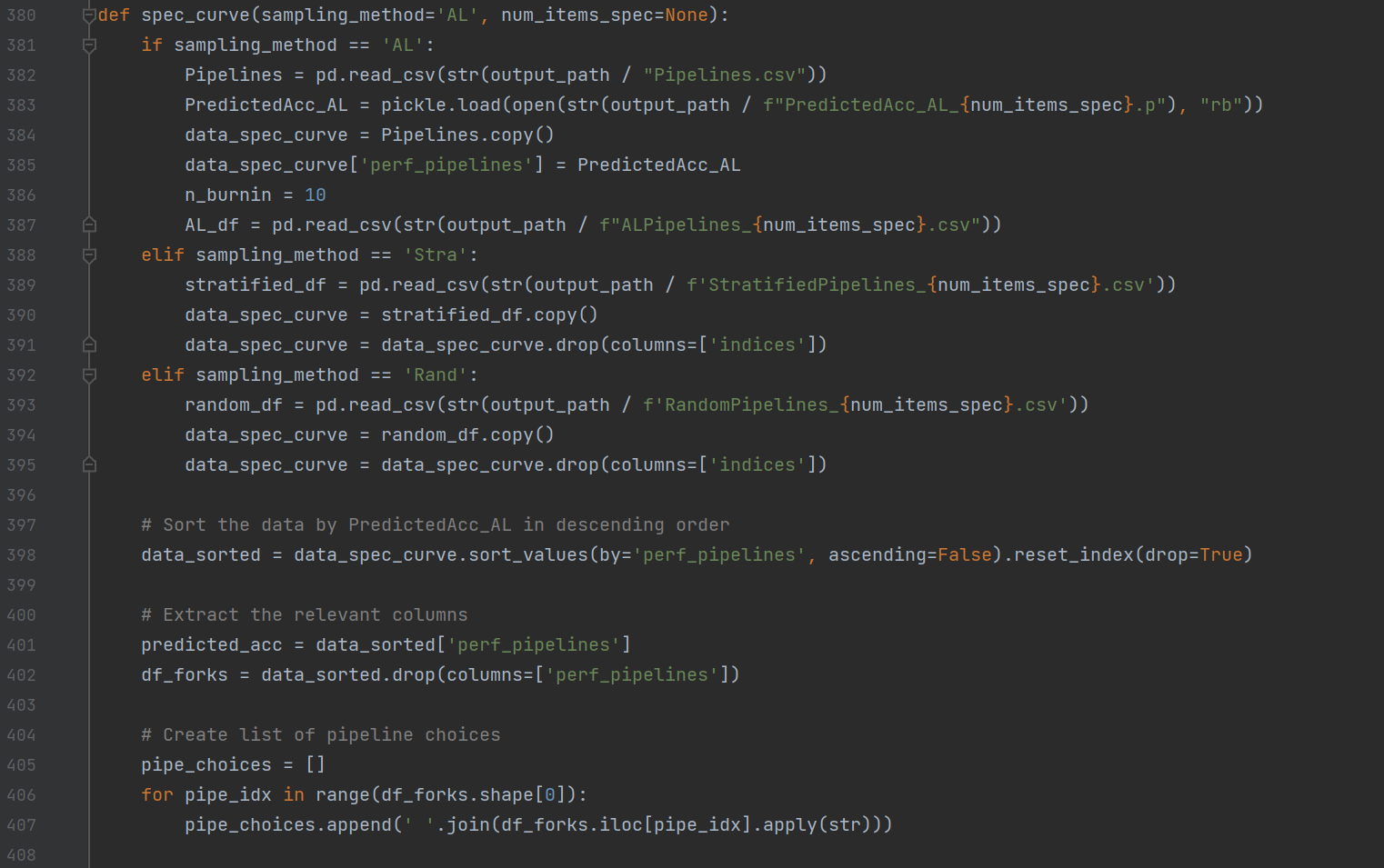
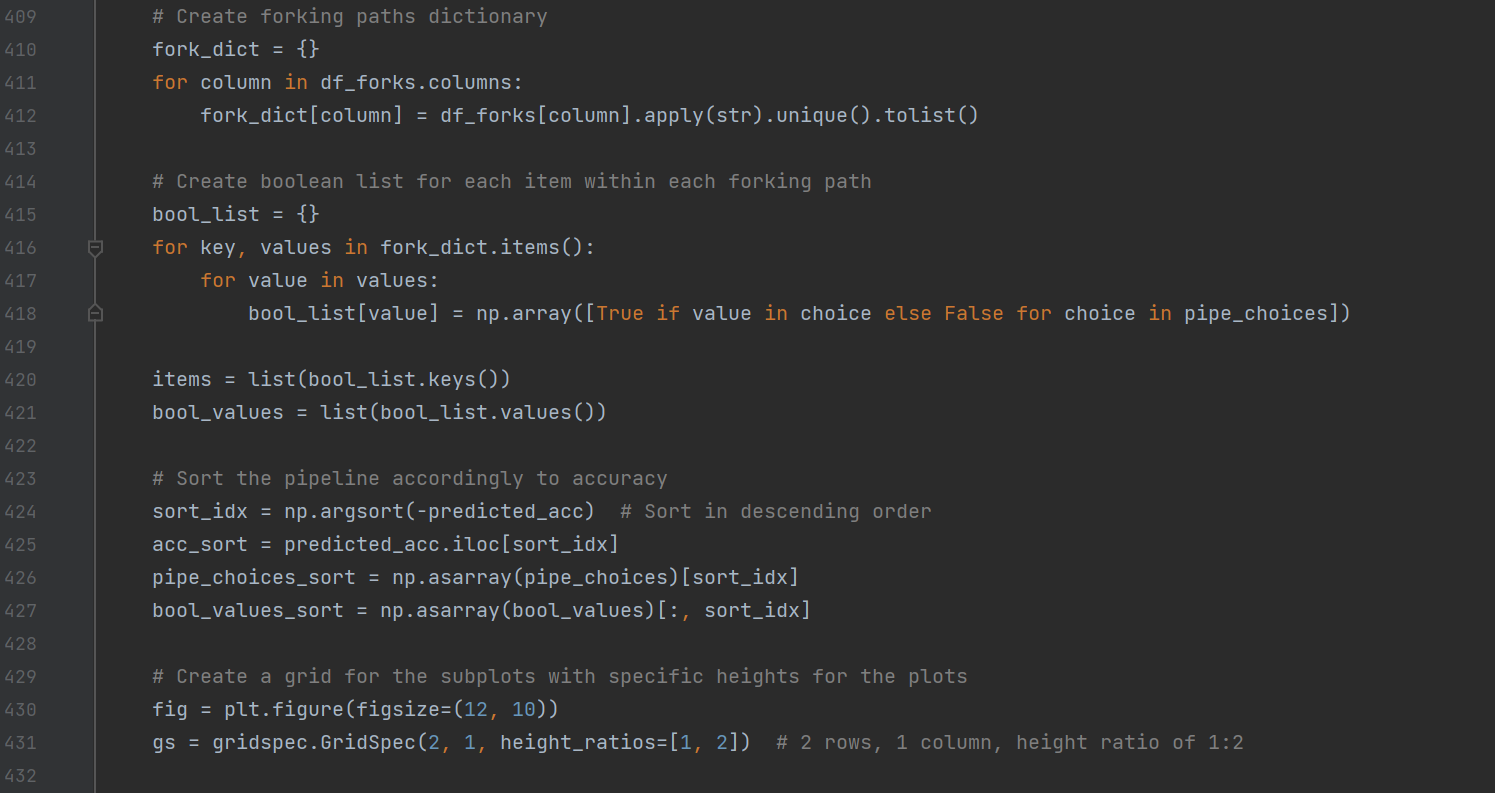
1. Initializes a dictionary (stratified\_samples) to store the stratified samples. Each key in this dictionary will represent a sample size, and each key corresponds to the stratified sample of that size;
2. Loops through the list of desired sample sizes;
3. For each sample size, generates a stratified sample using the create\_unique\_balanced\_stratified\_sample function, the current sample\_size, and the dataframe, and stores it in the stratified\_samples dictionary where all samples are organized and accessible by their size;
4. Returns the dictionary containing all of the stratified samples.

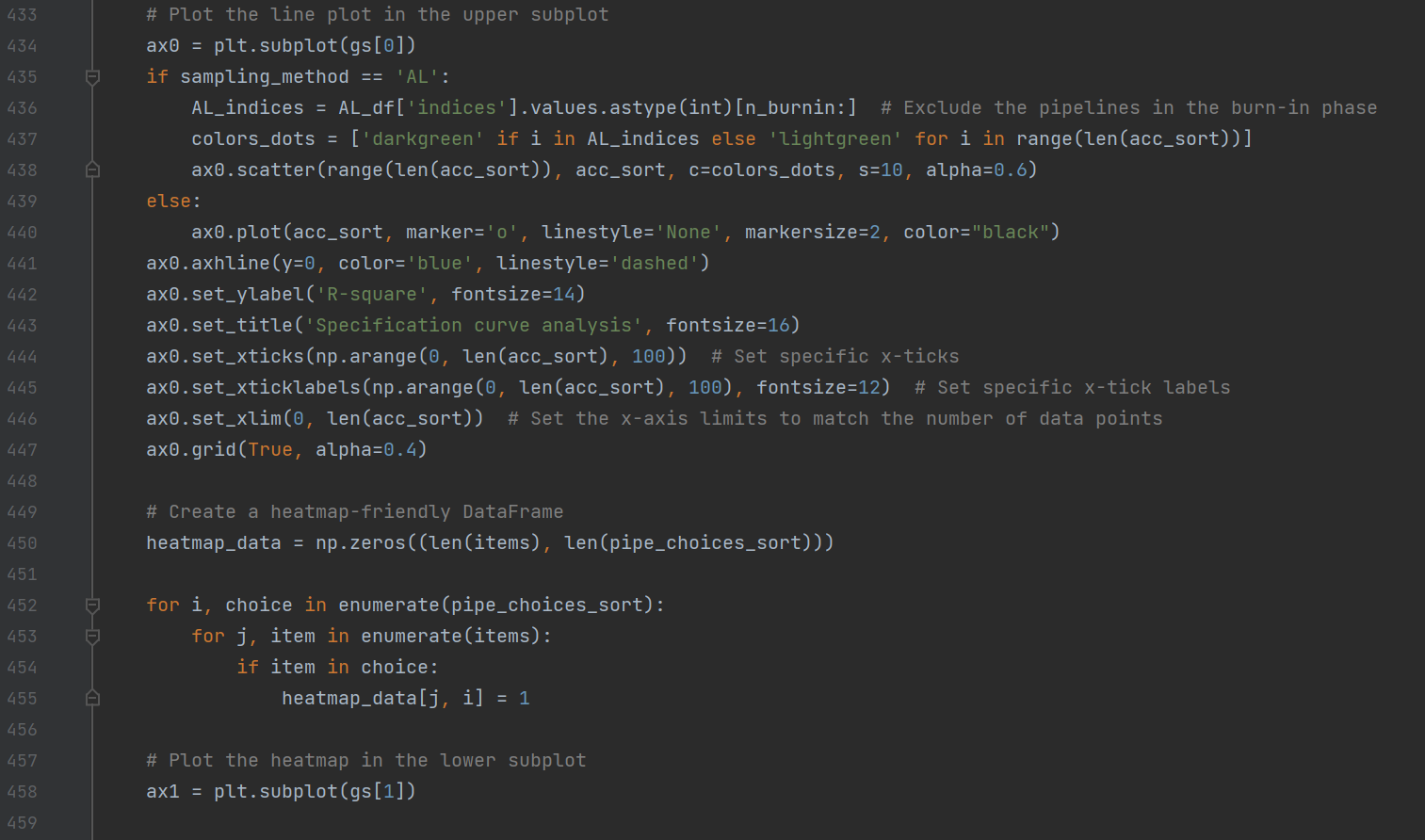
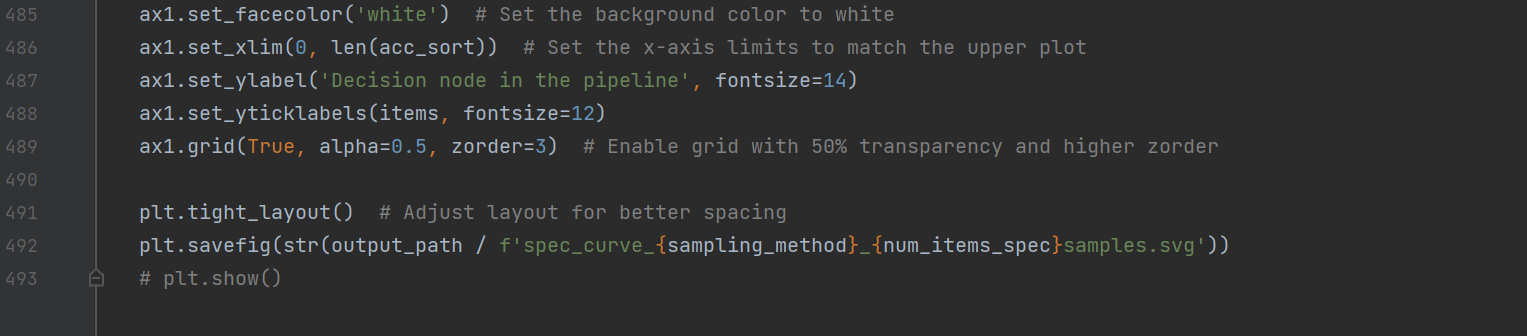
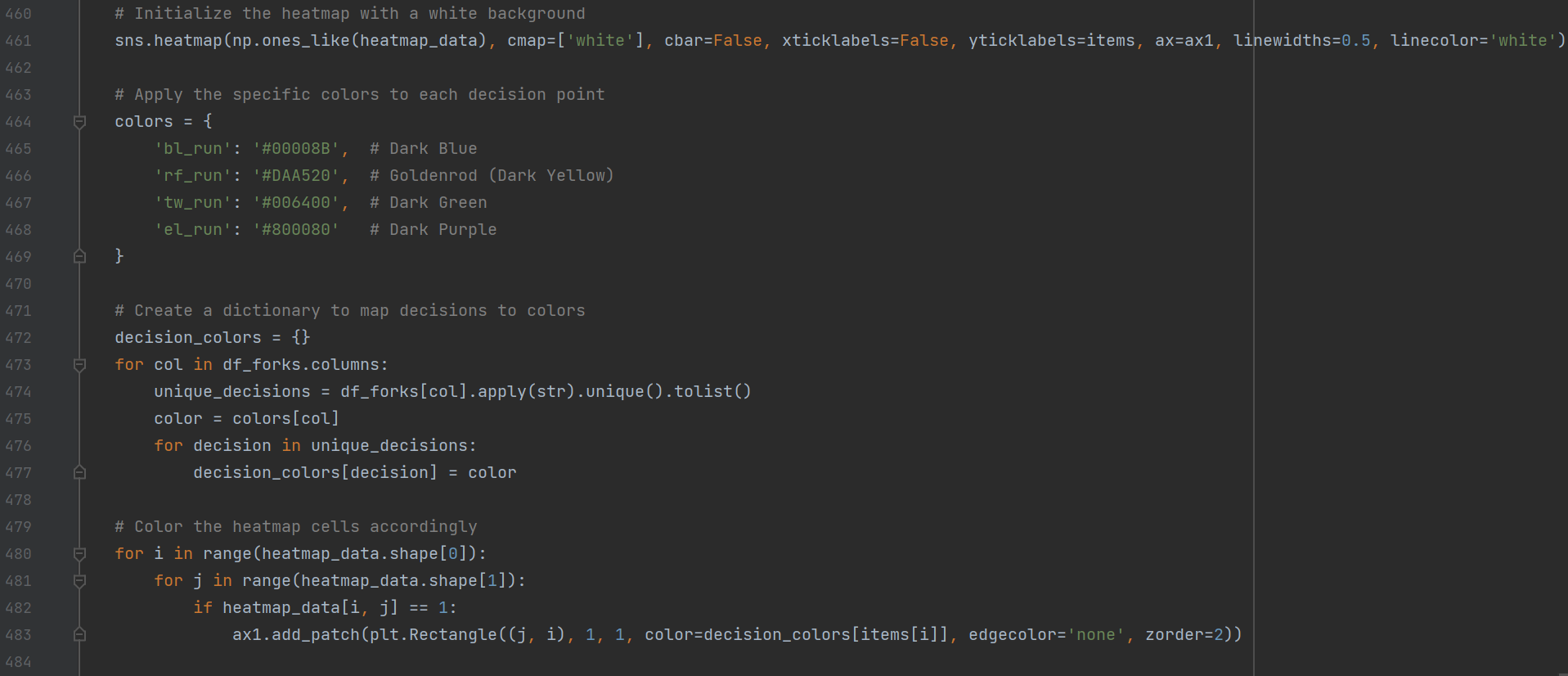
**Function average\_nearest\_neighbor\_distance**

This function calculates the average nearest neighbor distance between points in the sample and points in the reference:

1. Computes pairwise distances – the Euclidean distance between every point in the sample and every point in the reference. Distances is a two-dimensional array of shape (n\_sample, n\_reference), where each element represents the distance from a point in the sample to a point in the reference;
2. Calculates the nearest neighbor distance by computing the mean of the distances for each point in the sample across all points in the reference and taking the overall mean of these distances to get a single scalar value;
3. Returns the computed average nearest neighbor distance as a scalar to evaluate the proximity between the two sets of points.

**Function spec\_curve**





This function creates a specification curve figure, which consists of the upper panel that visualizes the performance of each pipeline and the lower panel that visualizes the options at each decision node that contribute to the respective pipelines of each result plotted in the upper panel.

1. Reads and processes pipeline performance data depending on the sampling\_method (AL, Stra, or Rand), and combines the pipeline decisions with performance metrics (perf\_pipelines) into a single dataframe. For AL (the active learning sample), it excludes the burn-in iterations (n\_burnin);
2. Sorts pipelines based on performance (descending order), and extracts decision paths and their unique combinations;
3. Creates a dictionary where each key is a decision node and the values are a list of unique options for that decision node;
4. Converts each option value into a Boolean array indicating whether that decision is present in each pipeline configuration;
5. Visualizes the results by plotting the upper and lower panels as described above. For AL (the active learning sample), the points in the upper panel are color coded to distinguish the pipelines that are sampled and the rest that are estimated;
6. Saves the visualization as an SVG file.