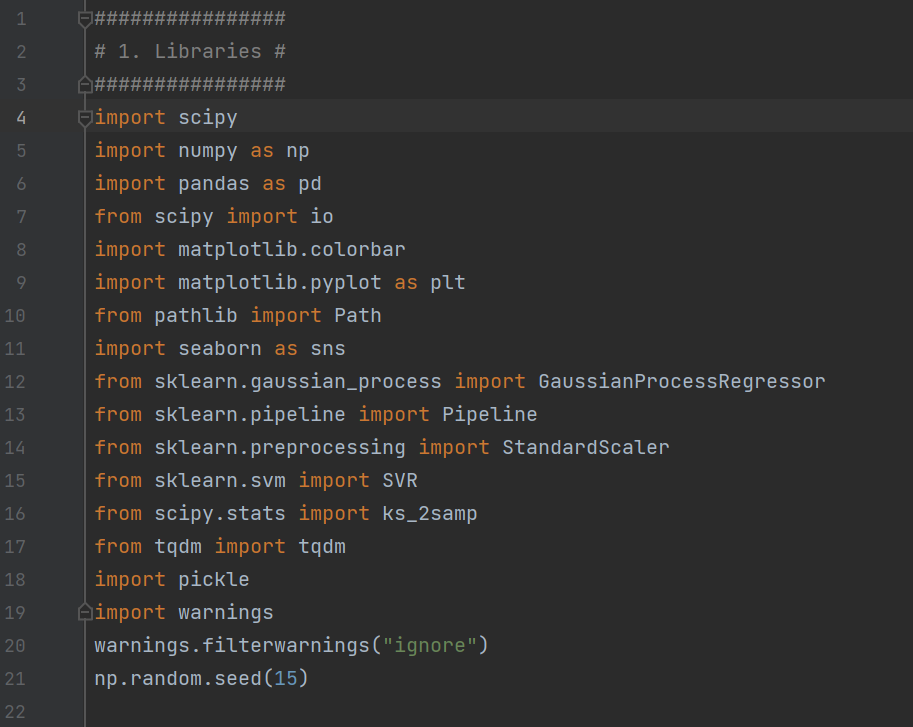
**Supplementary 1: The Structure and Modifiability of the Main.py Script**

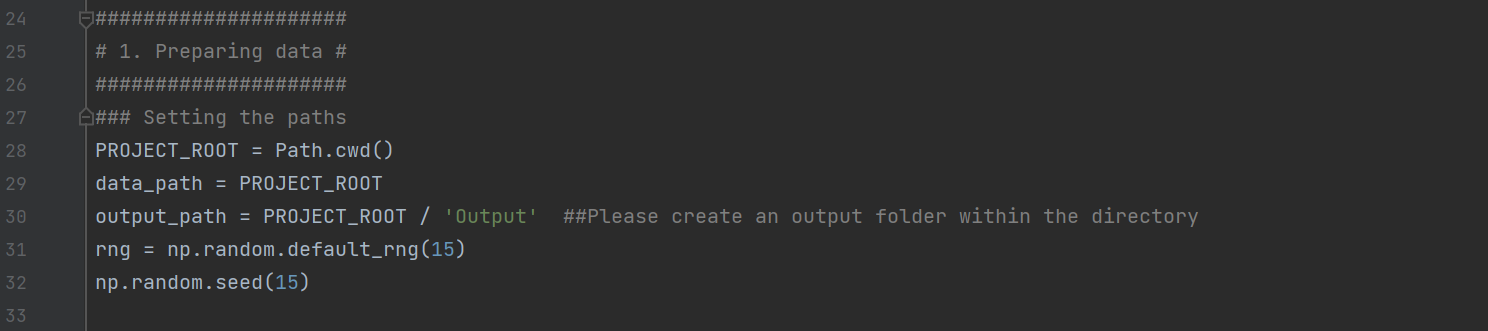
It is organized into five sections: preliminary installations, preparation, functions, sampling, and results.

**Section A: Preliminary Installations**

****

1. scipy, scipy import io, scipy-stats import ks\_2samp. This is a library that provides statistical functions used in the script. The ks\_2amp function from the scipy.stats module is used in the script to perform the Kolmogorov-Smirnov test.
2. numpy as np. In this script, NumPy is used for setting a seed for random number generation to ensure reproducibility and for performing random sampling.
3. pandas as pd. The panda library of data structures and analysis tools is used in the script to create and manipulate data frames, which are used to store and handle the configurations for different decision nodes and the results of the sampling and statistical tests.
4. matlabplot.colorbar, matplotlib.pyplot as plt. Matlabplot is a plotting library for creating visualisations. It's plotting functions are used in this script, for example for the scatter plots.
5. pathlib import Path. Path is used in the script to construct paths for data and output directories that is platform independent.
6. seaborn as sns. Seaborn is a statistical data visualisation library based on matplotlib. In this Python script, seaborn is used to create the raincloud plots.
7. sklearn.gaussian\_process import GaussianProcessRegressor, sklearn.pipeline import Pipeline, sklearn.preprocessing import StandardScaler, sklearn.svm import SVR. Th sklearn library is used for machine learning and contains simple and efficient tools for predictive data analysis. In this script, it's used for the Gaussian process regressor and the support vector regressor, which are part of the active learning sampling procedure.
8. tqdm. This is a library for creating progress bars for loops, and is used in this script to provide a visual reference for the progress of the sampling loops.
9. pickle. This Python module for serialising and deserialising object structures is used to save load previously saved data, for example. ‘PredictedAcc’.
10. Warnings, warnings.filterwarnings(“ignore”). This module controls the display of warnings, and this script uses it to suppress warnings that might clutter the output.
11. Np.random.seed(15) sets the seed for NumPy’s random number generator to 15. This ensures that any subsequent random operations produce reproducible results.

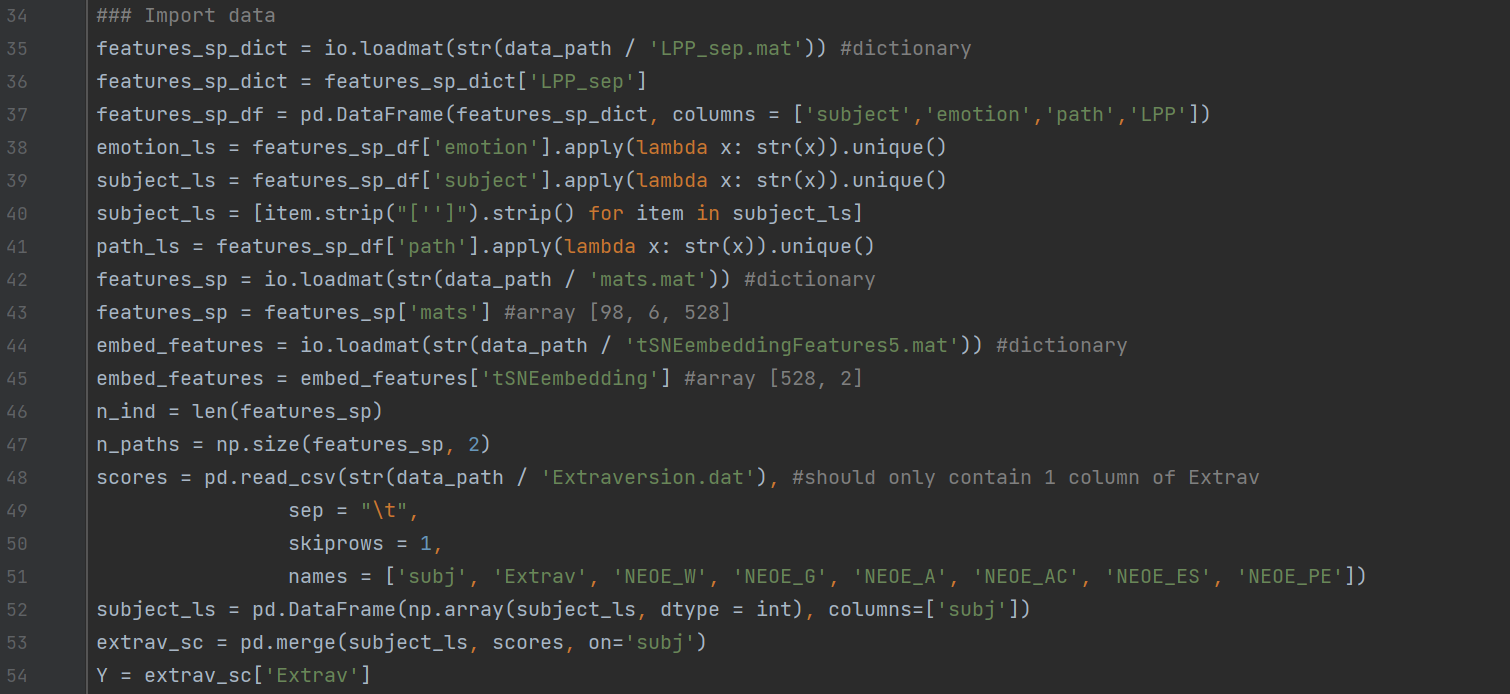
**Section B: Preparing data**



B1: Setting the paths

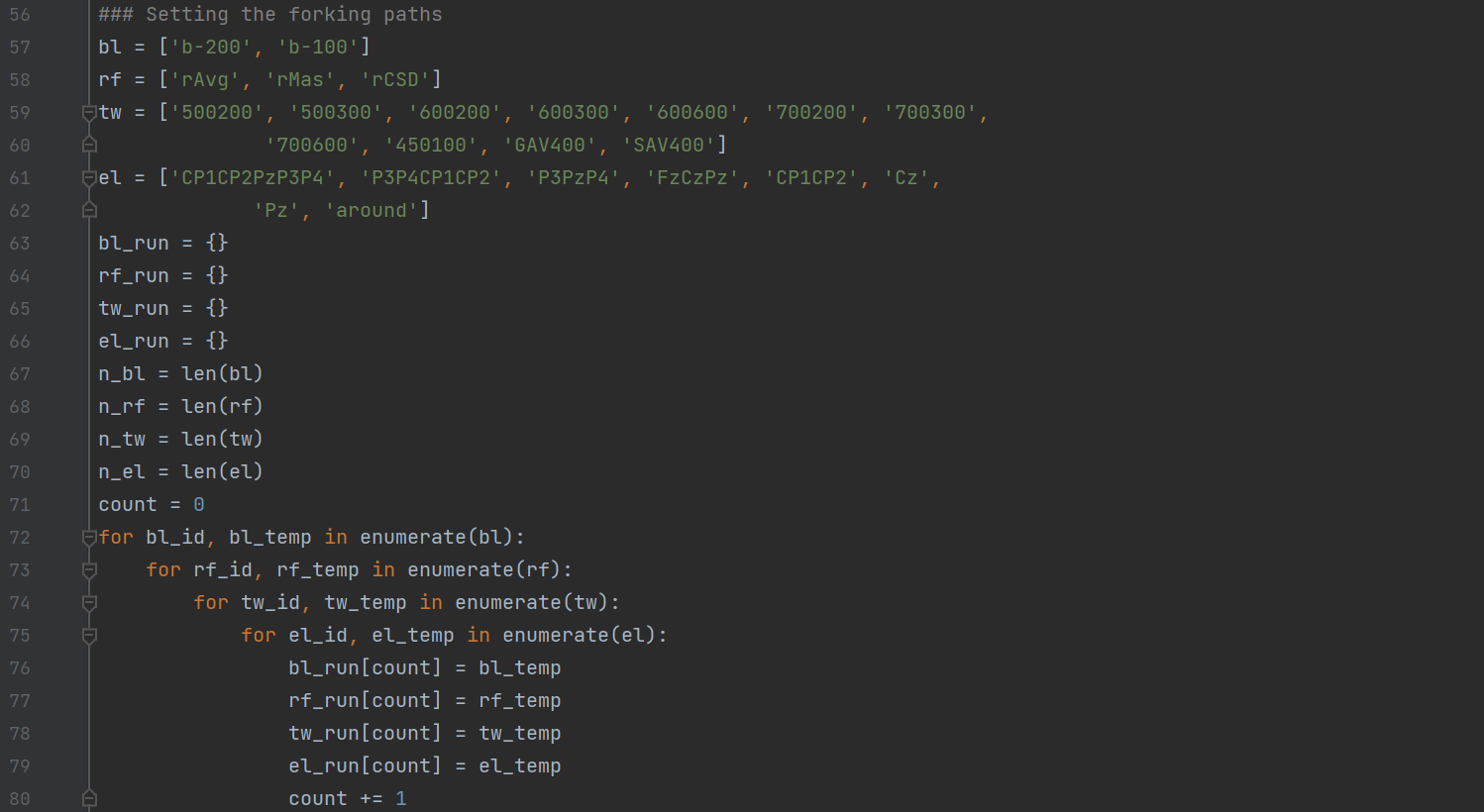
1. PROJECT\_ROOT = Path.cwd(). This sets the PROJECT\_ROOT variable to the current working directory using the cwd() function. The path entered should include the data files required for the script to run.
2. Data\_path = PROJECT\_ROOT. This assigns the PROJECT\_ROOT directory to the data\_path, as all data files should be located within the PROJECT\_ROOT directory that was set above.
3. Output\_path = PROJECT\_ROOT / ‘Output’. This creates an output path by appending an ‘Output’ directory to the PROJECT\_ROOT directory.
4. rng = np.random.default\_rng(11). This initializes a new random number generator object with a fixed seed (15) for reproducibility.
5. np.random.seed(15). Sets the seed for the random number generator to 15, ensuring reproducibility.

B2: Import data



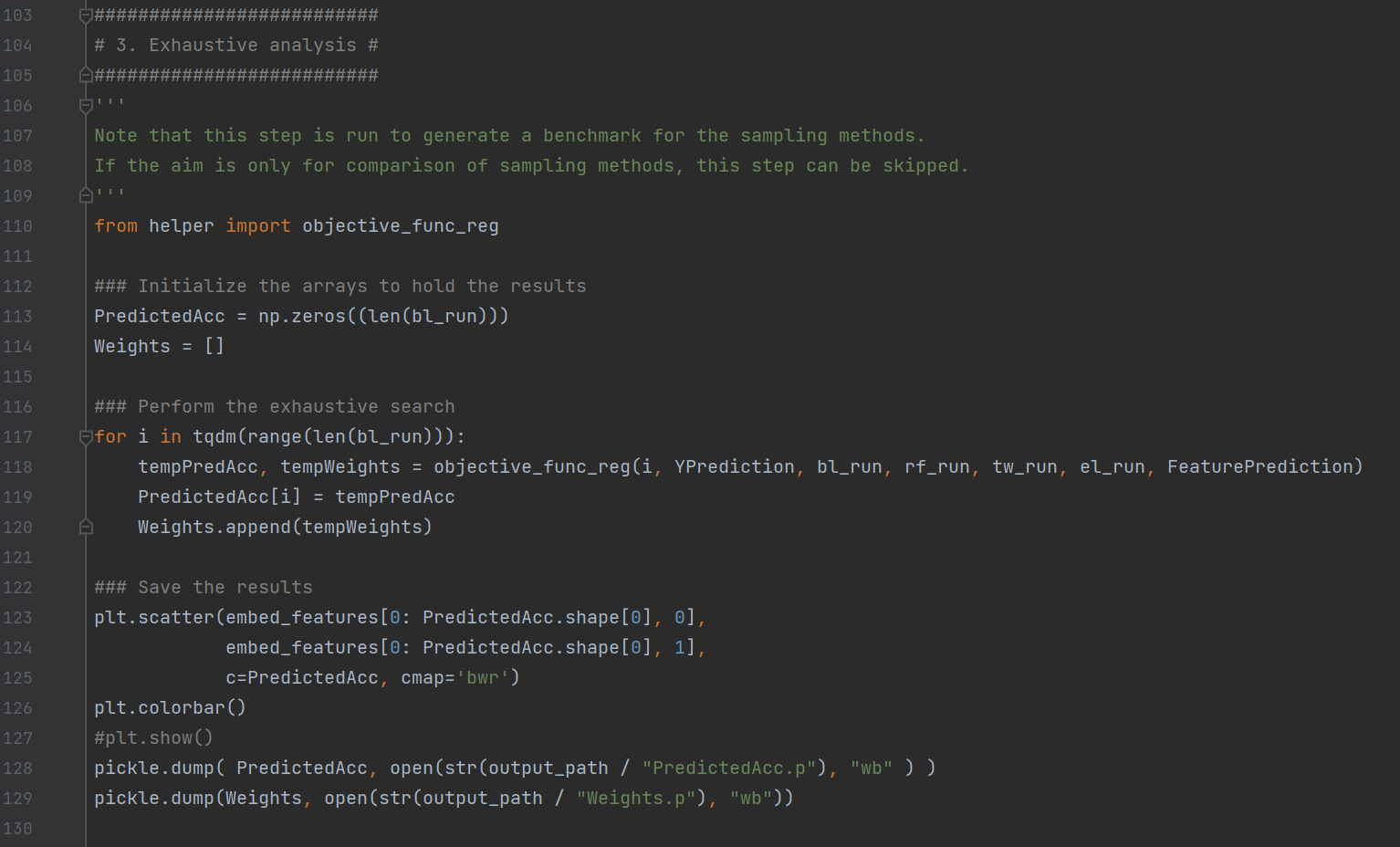
This subsection handles data importation, preparation and preliminary transformations:

1. Imports the LPP\_sep.mat file and specifies a dataframe (features\_sp\_df) with specific column names (subject, emotion, path, LPP);
2. Extracts unique values from the columns: emotion\_ls is a list of unique emotions, subject\_ls is a list of unique subjects (with formatting adjustments to remove extra characters), and path\_ls is a list of unique paths:
3. Loads mats.mat and specifies that it is an array (features\_sp) with shape [98, 6, 528] (98 participants, 6 conditions, 528 pipelines);
4. Loads tSNEembeddingFeatures5.mat and specifies it as an array with shape [528, 2] (528 data points embedded in a 2 dimensional space);
5. Stores dimensional information of features\_sp;
6. Loads extraversion scores from Extraversion.dat, which contains columns for participant ID, extraversion score, and additional personality scores that are not used in the current script, merges the data based on participant ID and extracts extraversion as the target variable.B3: Forked paths

B3: Setting the forking paths

1. Defines lists for different decision nodes of the multiverse analysis: electrodes, reference, time window and electrodes (bl, rf, tw, el). These should be replaced with names that represent the decision nodes of the user’s multiverse.
2. Initialises empty dictionaries for each of the decision nodes to store the generated configurations (bl\_run, rf\_run, tw\_run, el\_run). The prefix of these labels should be consistent with the labels you assigned in a for ease of understanding;
3. Counts and stores the number of options available for each decision node (n\_bl, n\_rf, n\_tw, n\_el; how many baselines, how many reference schemes, how many time windows and how many electrodes). The suffix of these labels should be consistent with the labels you assigned in sub-step ‘a’.
4. Nested for loop that iterates over each list (assigned in ‘a’), creating a Cartesian product of all configurations. Inside the loop, each configuration combination is assigned a unique count as its key and the corresponding values from each list are stored in their respective dictionaries initialized in ‘b’). The enumerate function is used to iterate over each list, providing the index (bl\_id, rf\_id, tw\_id, el\_id) and the value (bl\_temp, rf\_temp, tw\_temp, el\_temp). Again, the prefixes should be consistent with the labels assigned in ‘a’.
5. Converts four dictionaries (bl\_run, rf\_run, tw\_run, el\_run) into a dataframe, whre each row represents a unique pipeline configuration and saves the dataframe as a CSV file (Pipelines.csv).
6. Defines a cutoff point (20 participants in this case) for splitting the data (SpDefine). The data is partitioned into features for prediction (FeaturePrediction), which contains the data for participants after the first 20, and target for prediction (YPrediction), which contains the corresponding Y (extraversion scores) for these participants.
7. Optional: LockBox could define an additional subset for cross validation, RandomIndexes could randomly shuffle participant indices.

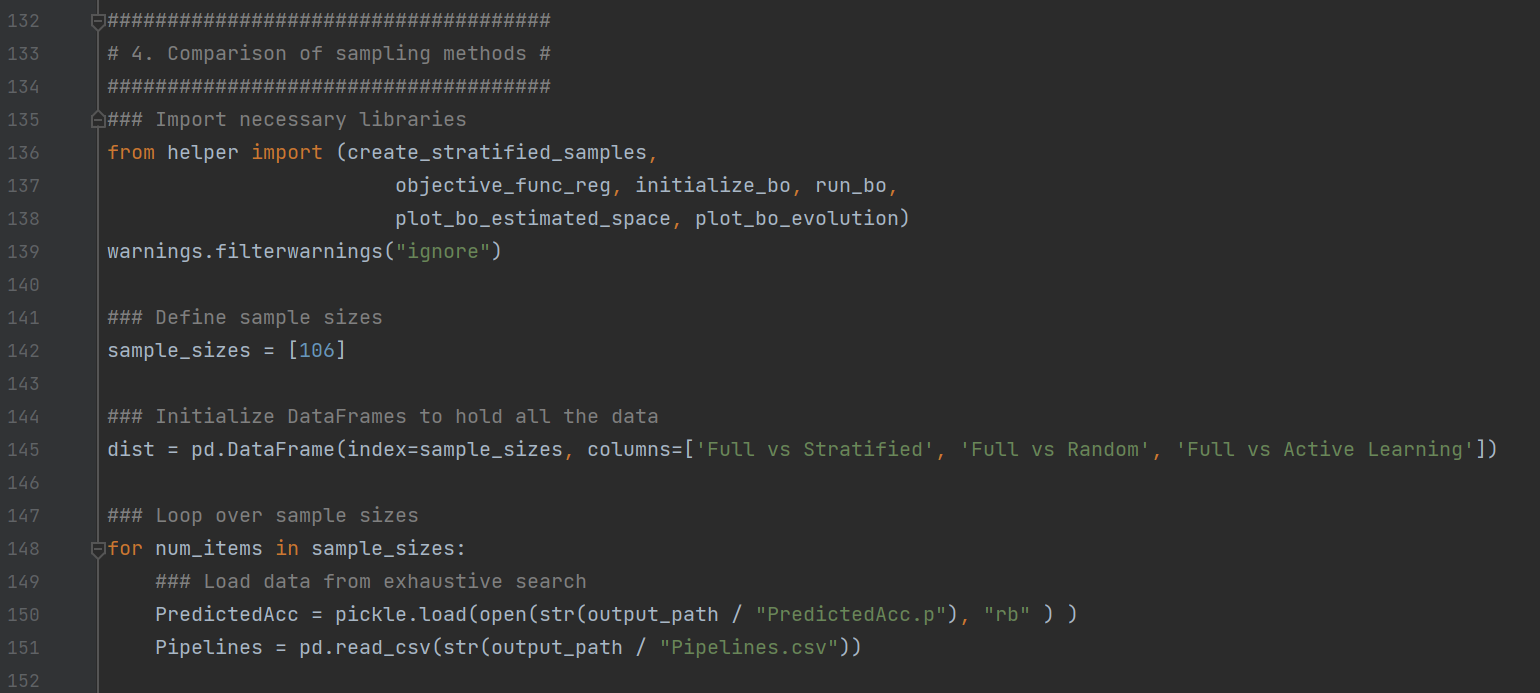
**Section C: Exhaustive analysis**



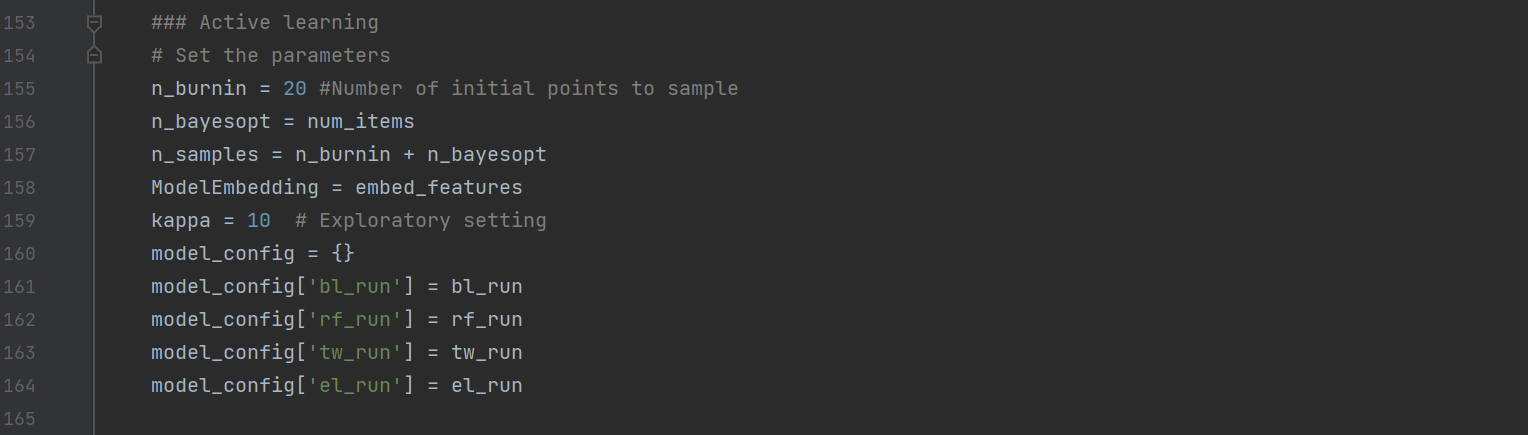
This section of the main script performs an exhaustive analysis to evaluate all pipeline configurations and generates a benchmark for later comparison with samples. This is not necessary and can be skipped when the multiverse is too large to compute exhaustively. This section calculates the performance (*R2*) for each pipeline and saves the results for later analysis.

1. Initializes arrays for results: PredictedAcc to store the performance (*R2*) for each pipeline, and Weights to store feature weights for each pipeline;
2. Performs the exhaustive search by iterating through all pipelines, calling the helper function objective\_func\_reg and imputing the pipeline index (i), the extraversion scores for prediction (YPrediction), the pipeline parameters (bl\_run, rf\_run, tw\_run, el\_run), and the features corresponding o the prediction subset (FeaturesPrediction). It outputs the *R2* score of the pipeline (tempPredAcc) and the feature weights from the linear regression model (tempWeights), and stores the results in PredictedAcc and Weights;
3. Creates a scatterplot of the distribution of pipeline performance across the two-dimensional space;
4. Saves the results.

**Section D: Comparison of sampling methods**

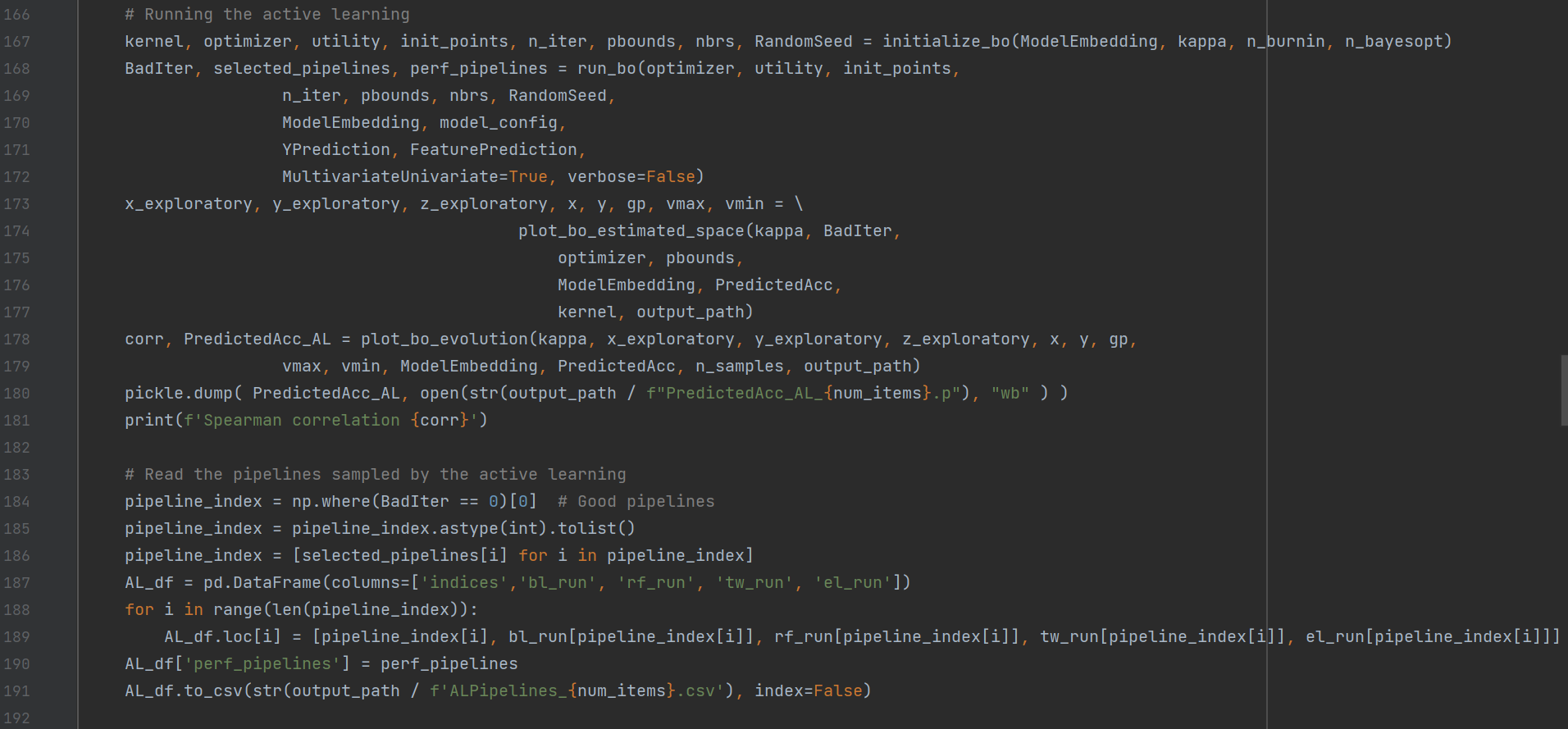
Section D is a large code chunk that computes and compares the active learning, stratified and random samples. This section is explained in smaller sub-chunks.

1. Imports key functions from the helper script;
2. Defines sample sizes (the example in the code image is 106, which means that each sampling method will select 106 pipelines - 20% of the 528 - for evaluation);
3. Creates an empty dataframe (dist) to store similarity metrics between the full exhaustive benchmark and the sampled subsets. The columns represent comparisons of the exhaustive multiverse to the stratified sample, the random sample and the active learning sample;
4. Initiates a loop to iterate over the defined sample sizes;
5. Loads the previously computed benchmark data from the exhaustive computation, where PredictedAcc contains the *R2* for all pipelines, and Pipelines is a dataframe listing all pipeline configurations;

****

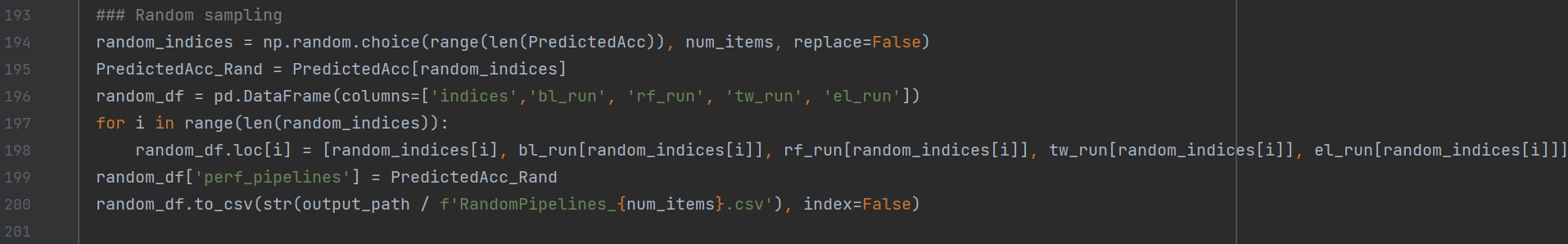
This sub-chunk prepares the parameters and configurations for implementing active learning.

1. Sets the active learning parameters: n\_burnin (the number of randomly selected pipelines to evaluate before starting Bayesian optimization), n\_bayes\_opt (the number of additional pipeline evaluations to perform using Bayesian optimization), n\_samples (total number of pipelines to evaluate, which is n\_burnin + n\_bayesopt);
2. Assigns the precomputed t-SNE embeddings (embed\_features) of pipelines to ModelEmbedding;
3. Sets the exploration vs. exploitation trade-off parameter for Bayesian optimization. A higher kappa value, 10 as cited by Dafflon et al., (2022) provides a more explorative sample, favoring the selection of data points with greater uncertainty, whereas a lower kappa value, such as 0.1 as cited by Dafflon et al., (2022) provides a more exploitative sample, favoring data points that are predicted to improve the model performance based on current knowledge of the sample so far;
4. Creates a dictionary (model\_config) to map pipeline components to their respective configurations (bl\_run, rf\_run, tw\_run, el\_run), which provides a structured way to pass all pipeline configurations to the Bayesian optimization process.

As with the other parameters set in the script, the user can adjust these parameters before running the script based on their data and multiverse analysis goals. We would like to reiterate that the active learning sampling feature in the Python script is a modification of that published by Dafflon et al. (2022), available [here](https://github.com/Mind-the-Pineapple/into-the-multiverse).

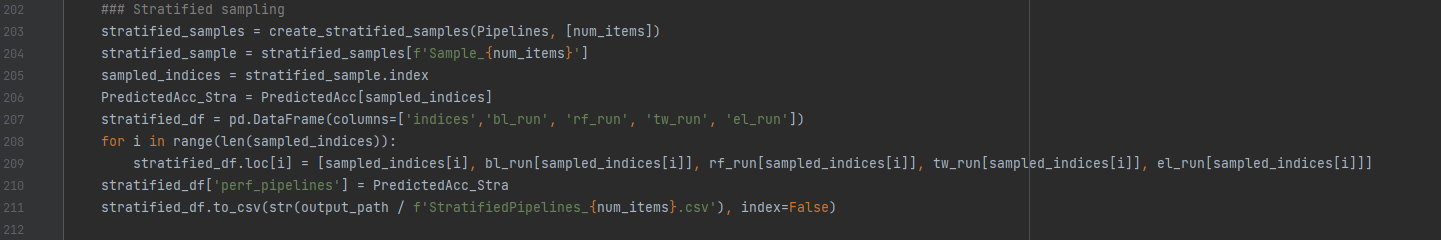
This sub-chunk executes the active learning sampling. It includes steps for initializing the Bayesian optimization, running the optimization, visualizing the results, and saving the sampled pipelines and their performance:

1. Calls the initialize\_bo function from the helper script to set up the Bayesian optimization with the following outputs:
   1. Gaussian process regressor kernel: In the use case, a combination between a matern and a white noise kernel is used, which was default in the script as downloaded from Dafflon et al., (2022). The default can be adapted if the user wishes to modify the smoothness and the noise variance;
   2. Optimizer object;
   3. Utility function for selecting the next point in the space;
   4. Number of random initialization points (burn in sample);
   5. Number of iterations for the Bayesian optimization;
   6. Parameter bounds for the optimization process;
   7. Nearest neighbor model for pipeline embeddings;
   8. Seed for reproducibility.
2. Executes the Bayesian optimization process using the run\_bo function, which selects pipelines iteratively based on the utility function and evaluates selected pipelines using the objective\_func\_reg function;
3. Visualizes the search space using the plot\_bo\_estimated\_space function;
4. Visualises the Bayesian optimization evolution using the plot\_bo\_evolution function;
5. Saves the results and the sampled pipelines (identified pipelines selected with acceptable performance - BadIter == 0, creates a dataframe (AL\_df) to store the indices of selected pipelines, pipeline configurations, and performance scores).

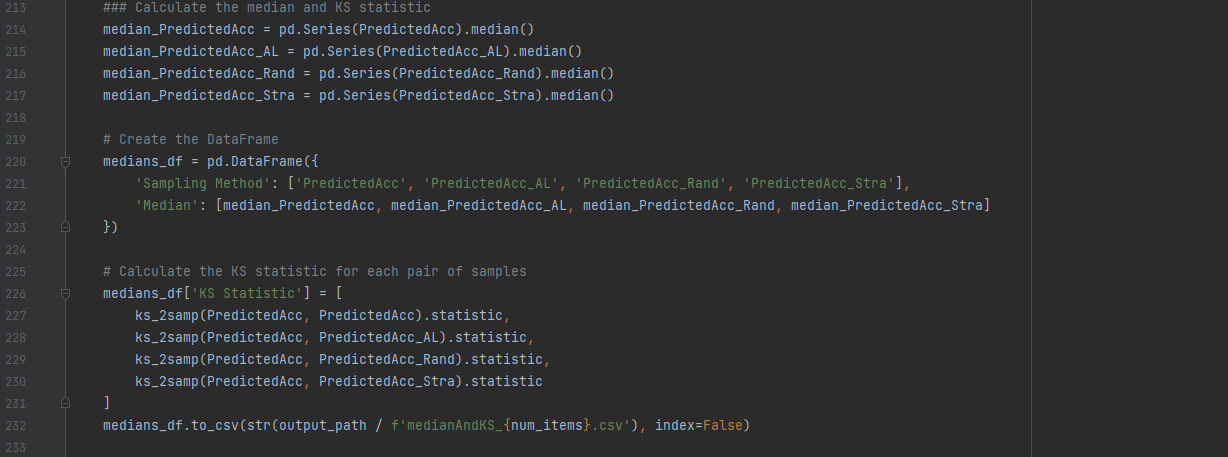
****

This sub-chunk implements random sampling, where pipelines are selected randomly, and evaluates the performance of the randomly sampled pipelines:

1. Randomly selects num\_items unique pipeline indices from the full set of pipelines, ensuring that no index is selected more than once;
2. Extracts the performance scores (PredictedAcc) of the randomly selected pipelines (random\_indices);
3. Creates a dataframe (random\_df) for the randomly selected pipelines with columns to store indices and corresponding pipeline cofigurations (bl\_run, rf\_run, tw\_run, el\_run), and fills the dataframe with the information of the sampled pipelines;
4. Adds a column to the dataframe that contains the performance scores (PredictedAcc\_Rand);
5. Saves the dataframe as a CSV file.

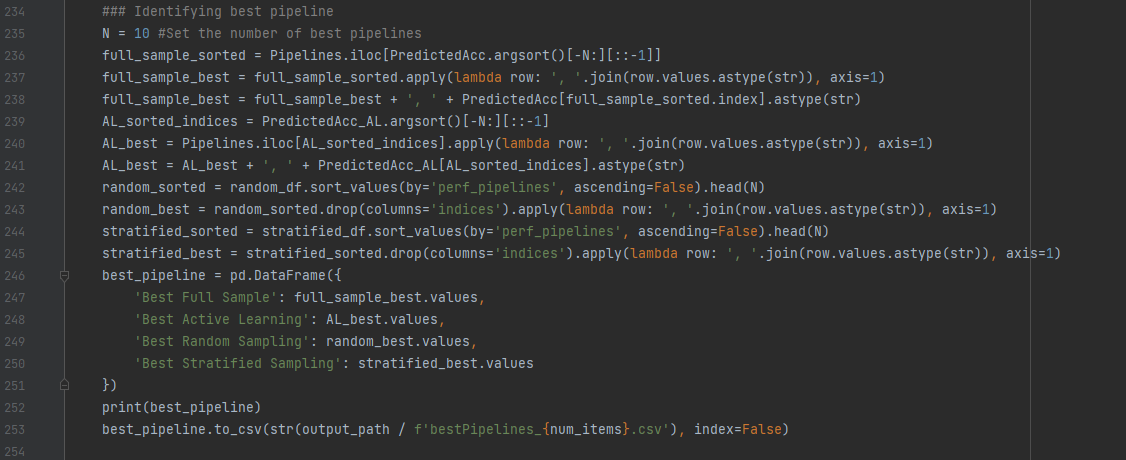
****

This sub-chunk implements stratified sampling by selecting pipelines proportionally to represent the diversity of the entire identified multiverse, and evaluates their performance:

1. Uses the create\_stratified\_samples function from the helper script with inputs of the dataframe that contains all pipelines (Pipelines), and the list of desired sample sizes (num\_items). It outputs a dictionary (stratified\_samples) where each key corresponds to a sample size;
2. Collects the stratified sample corresponding to num\_items, and extracts the indices of the sampled pipelines (sampled\_indices);
3. Retrieves the performance scores (PredictedAcc) of the pipelines in the stratified sample using their indices;
4. Initializes an empty dataframe (stratified\_df) with columns for the indices of sampled pipelines and the pipeline configurations, and fills the dataframe with the information about the sampled pipelines;
5. Adds a column (perf\_pipelines) to the dataframe containing he performance scores (PredictedAcc\_Stra) of the pipelines in the sample;
6. Saves the dataframe of the sampled pipelines.

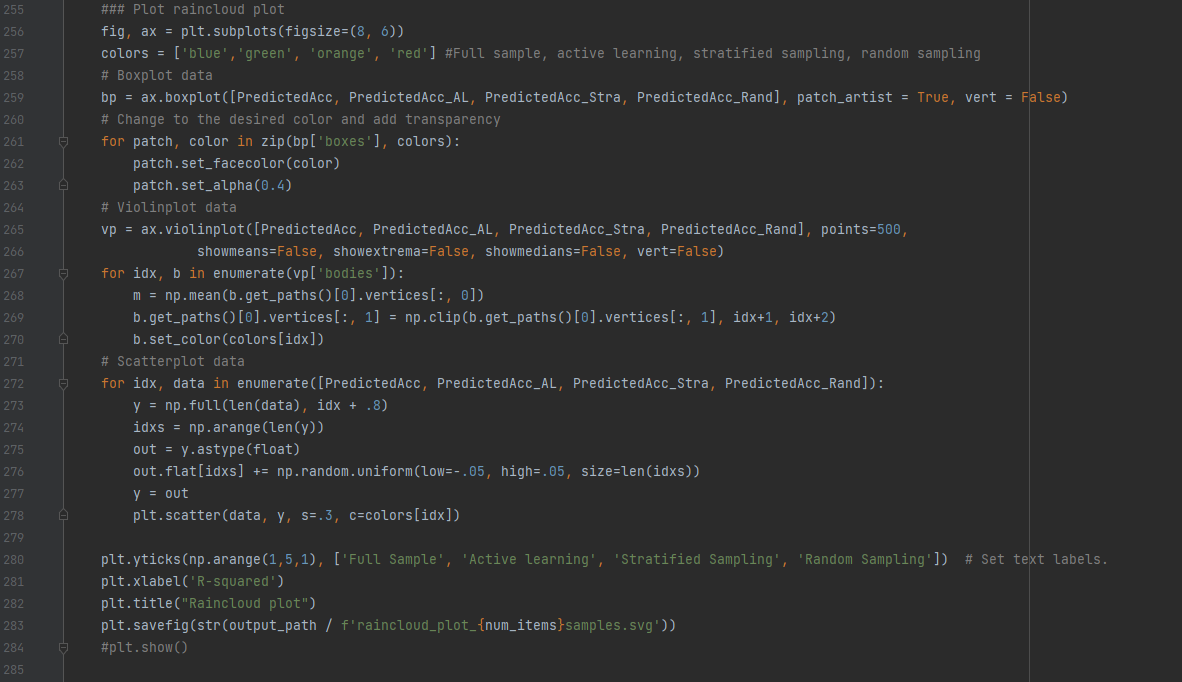
This sub-chunk computes and compares the median performance and the Kolmogorov-Smirnov (K-S) statistic for pipeline performance across different samples:

1. Calculates the median performance for the full multiverse if this is computed, the active learning sample, random sample, and stratified sample;
2. Creates a dataframe to store the median performance for each sample;
3. Calculates the K-S statistic for each sample vs. the full multiverse;
4. Saves the results.



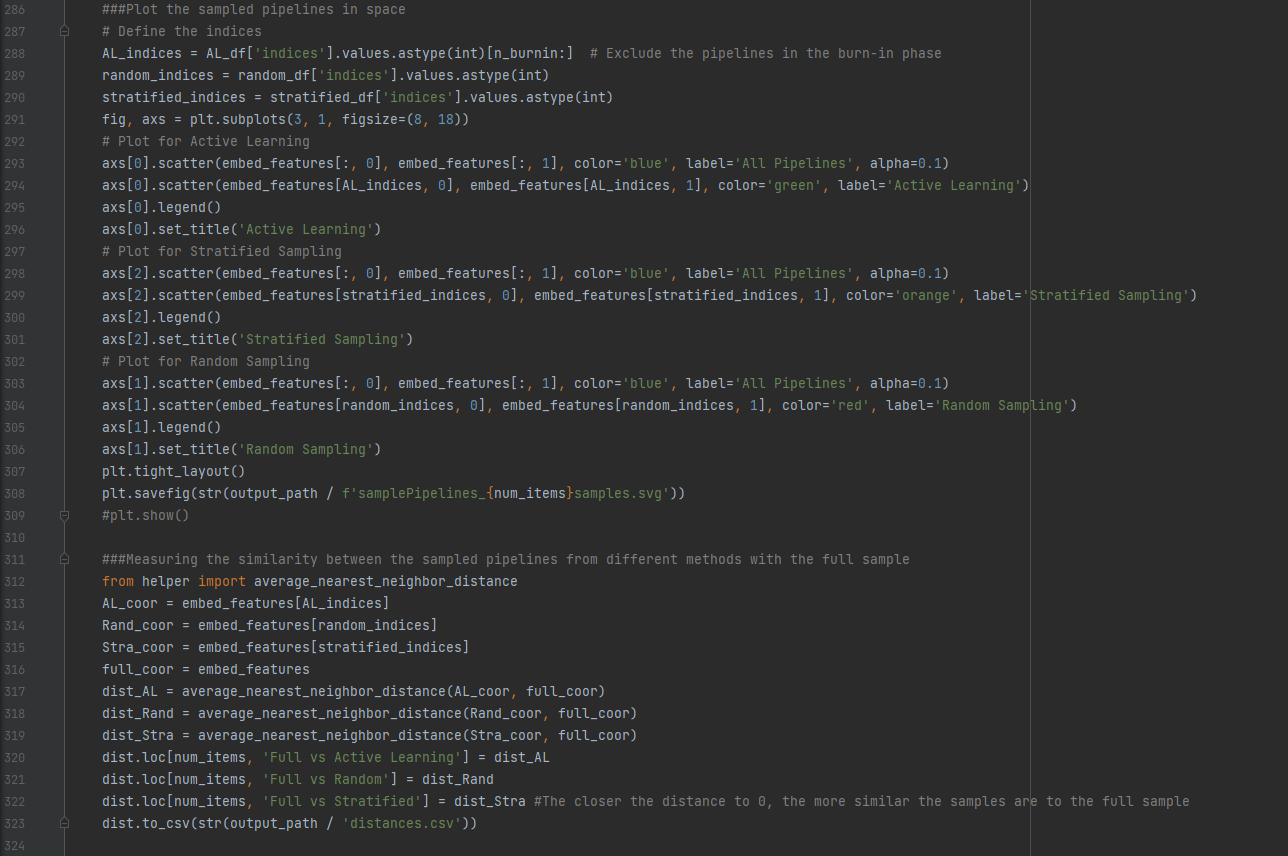
This sub-chunk identifies and compares the top performing pipelines for the full multiverse and each of the three samples:

1. Specifies the number of pipelines to list (e.g., top 10 performing pipelines for each);
2. Extracts the top performing pipelines from the full multiverse and each sample by sorting the indices of the PreditedAcc array in ascending order, selecting the top N indices in descending order of performance, retrieving the pipeline configurations corresponding to these, converting each pipeline into a comma-separated string, and appending the performance scores to the pipeline strings;
3. Creates a dataframe of the top performing pipelines for the full multiverse and each sample with columns for the full multiverse and each sample;
4. Saves the results.

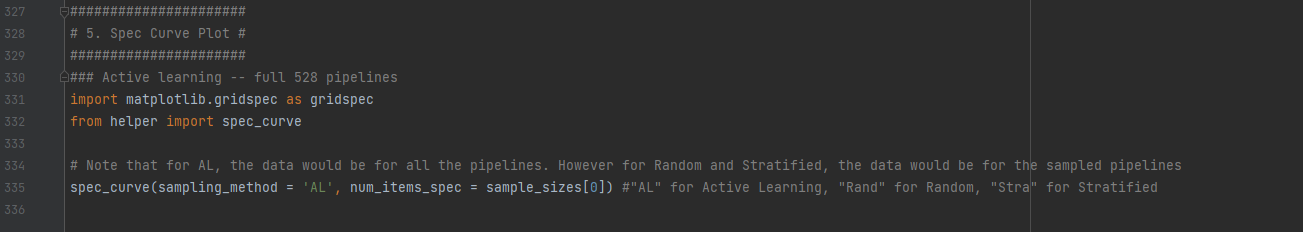


This sub-chunk visualizes the performance distribution of pipelines sampled by the different methods using a raincloud plot:

1. Initializes a plot with specified figure size and single axis and defines the color palette for the sampling methods;
2. Creates a horizontal boxplot for the distributions of the PredictedAcc for the full multiverse and each of the samples;
3. Loops through the boxplot patches and assigns color and transparency;
4. Creates a horizontal violin plot for the same distributions as the boxplot and iterates through the violin plot to assign color and alignment adjustment;
5. Adds jittered scatter points to represent individual data points in each distribution;
6. Sets the axis labels;
7. Saves the plot as an SVG file.



This sub-chunk visualizes how pipelines selected by the different sampling methods are distributed in the two-dimensional space relative to the full multiverse, and calculates similarity metrics between the sampled pipelines and the full multiverse:

1. Extracts indices of pipelines sampled by each method (excluding the burn-in pipelines for the active learning sample);
2. Creates a figure with three subplots stacked vertically (one subplot for each sample), specifying the figure size;
3. For each subplot, all pipelines are plotted in the two-dimensional space as faint blue points, and the sampled pipelines are colored as green for the active learning sample, orange for the stratified sample, and red for the random sample;
4. Saves the plot.
5. Measures the similarity of each ample with the full sample by extracting coordinates of sampled pipelines and all pipelines from the embedding space, uses the average\_nearest\_neighbor\_distance function from the helper script to compute the similarity of each sample of pipelines to the full multiverse;
6. Saves the results.

This sub-chunk generates a specification curve plot to visualize the performance of pipelines across different sampling methods. Uses the spec\_curve function to create the specification curve, with AL entered for active learning, Rand entered for random, and Stra entered for stratified.