**Lead Gen(V2) Documentation** -

1. **Column\_mappingv2.py**

Lead Generation Code Initialization:

* + Begins with running the columns\_mappingv2.py script.
  + Accepts three input arguments: file\_path (str, required), industry\_profile (str, optional), and feature\_columns (str, optional).
* File Processing:
  + Reads the specified file and generates an output filename by appending '-columnmappingv2'.
  + Cleanses the file using the basic\_preprocessing function from utils:
    - Converts column names to lowercase, and replaces spaces with underscores.
    - Substitutes '/' with underscores.
    - (To do) Replaces non-alphanumeric characters with underscores.
    - Standardizes column names.
    - Removes rows with identical or null values.
    - Eliminates duplicate columns.
* Dataframe Augmentation:
  + Applies the Create\_columns\_if\_doesnt\_exist function (from utils) to add missing columns:
    - 'full\_name', derived from 'first\_name + last\_name', 'first\_name', or 'last\_name'.
    - 'full\_address', created from 'street\_address + state + zipcode'.
    - 'owner\_full\_name', formed from 'owner\_first\_name + owner\_last\_name', 'owner\_mailing\_name', or 'owner\_first\_name'.
    - 'lot\_area\_sqft', generated if 'lot\_area\_sqft' and 'lot\_acreage' are absent.
    - Mail and property state abbreviations are obtained using 'abbreviate\_state\_names'.
    - 'property\_zip\_code' extracted from 'property\_address\_full' or 'ZIP-COUNTY-FIPS\_2017-06.csv'.
* Address Verification:
  + Checks completeness of 'mail\_full\_address' using a lambda function, 'is\_full\_address'.
  + Renames 'mail\_full\_address' to 'mail\_street\_address' if 50% of addresses are incomplete.
* Additional Columns Creation:
  + Generates 'mail\_zip\_code' (if absent) from 'mail\_full\_address' or 'owner\_city\_state\_zip'.
  + Creates 'mail\_cty' and 'mail\_state\_name\_short\_code' from 'mail\_city\_state'.
* Column Validation:
  + Uses the 'required\_columns\_exist' function to ensure minimum required columns ('full\_name', 'zipcode', 'full\_address', and 'feature\_columns').
  + Raises an exception if any required columns are missing.
* Data Cleaning:
  + Applies cleaning functions ('clean\_email', 'clean\_name', 'clean\_zipcode') to ensure data adheres to specified patterns.
* Logging and Output:
  + Logs success status ('True') upon completion.
  + Writes the dataframe to a new output file with the extension '-columnmappingv2.csv'.

1. **preprocessv2.py**\*\*Preprocessing with preprocessv2.py:\*\*
   * - Executed after successful completion of `columnmappingv2.py`.
   * - Requires four arguments:
   * - `filepath` (required, type=str): File ending with '-columnmappingv2'.
   * - `Max\_rows` (required=false, type=int): Maximum rows needed for model creation or displaying results.
   * - `feature\_columns` (required=false, type=str): Columns specified by the client for featurization.
   * - `Data\_type` (required=false, type=str, default='train'): Defines whether it's a train or test pipeline (TODO: Not currently in use).
   * - `Industry\_profile` (required=false, type=str): Specifies the type of industry.

\*\*Output File Path:\*\*

* + - Creates the output file path by appending '-preprocessv2.csv' as an extension.

\*\*Data Sampling with 'select\_data' Function:\*\*

* + - Applies the `select\_data` function to the dataframe.
  + - Calculates the `value\_counts` (weightage) of rows based on the frequency of zip codes.
  + - Randomly selects a subset of rows (min(num of rows, 1000)) while maintaining the same weightage.
  + - Aims to create/run the model on a manageable subsample when dealing with large client-uploaded files.

\*\*Data Appending with 'append\_data' Function:\*\*

* + - Utilizes the `append\_data` function.
  + - Uses Melissa API to append data, including required columns: ["DateOfBirth", "Education", "Gender", "HouseholdIncome"].
  + - Allows passing extra columns from arguments (e.g., [OwnRent, PoliticalParty, PresenceOfChildren, TypeOfVehicles]).
  + - Prints the time taken after processing every 500 rows.
  + - Replaces null values with empty strings for the appended data.

\*\*Age Calculation with 'calculate\_age' Function:\*\*

* + - Applies the `calculate\_age` function to determine the 'age' using the 'dateofbirth' column.

\*\*Output and Logging:\*\*

* + - If all the above steps run successfully, write the output to a CSV file.
  + - Logs the success with information: {"Success": "True"}.

1. **model\_creationv2.py**

* Input Arguments:
  + Takes a single argument: file\_path (type=str, req=True).
  + Requires the filepath to end with ‘-preprocessv2.csv’.
* Output File from model\_creationv2.py:
  + The output file is a CSV file with the extension ‘-modelcreationv2.csv’.
* Additional Parameters:
  + Passes constant\_parquet\_path (cluster\_zipcode\_mapping.json) to store zipcodes and their respective clusters.
  + Stores 'processed\_zipcode\_data.parquet' in zipcode\_cluster\_mapping\_df.
* Traffic Density Calculation:
  + Applies the get\_zipcode\_density function to the preprocessed file (individual\_df).
  + Calculates traffic density, i.e., sales/conversion/purchases per zipcode compared to the total dataset.
  + The output provides columns 'zipcode' and 'count density', stored in zipcode\_density\_df.
* Distance Matrix Data:
  + Reads an 8GB pre-built 'distance\_matrix\_data' containing the distance matrix (41k X 41k) between all zipcodes.
* Probability Calculation:
  + Calls the calculating\_probability\_per\_zipcode function:
    - Takes inputs: zipcode\_cluster\_mapping\_df, zipcode\_density\_df, distance\_matrix\_data, and cluster\_info.
    - Calculates the probability of conversion for each zipcode using the formula: probability += 1/Di \* exp(count\_density(i)).
    - For each row in a cluster (C), sums the formula using the distance (Di) between that row and each row in the cluster (D), multiplied by the count density of each row in the cluster.
    - Zipcodes with no count density contribute to the formula with 1/Di.
    - For zipcodes with a distance of 0, the second nearest distance is considered.
  + Outputs stored in zipcode\_probability\_df.
* Cleanup:
  + Runs the garbage collector and deletes space-consuming Parquet files.
* Data Merging and Handling:
  + Merges zipcode\_probability\_df on zipcode\_cluster\_mapping\_df.
  + Fills empty 'conversion\_probability' with median probability values.
* Sorting and Output:
  + Sorts zipcodes based on 'conversion\_probability' (high to low).
  + Writes the output to a file with an extension of ‘-modelcreationv2.csv’.
* **4. Criteria\_generationv2.py**
* #### 1. Importing Libraries:
* - The code begins by importing several Python libraries, including:
* - `pandas` (as `pd`) for data manipulation and analysis.
* - `re` for regular expressions.
* - `numpy` (as `np`) for numerical operations.
* - `math` for mathematical functions.
* - `os` for interacting with the operating system.
* - `json` for JSON data handling.
* - `requests` for making HTTP requests.
* - `time` for time-related operations.
* - `logging` for logging messages.
* - `argparse` for parsing command-line arguments.
* - `datetime` for working with dates and times.
* #### 2. Configuration of Logging:
* - Logging is configured with a basic configuration using `logging.basicConfig`. The logging level is set to INFO, and the format is specified.
* #### 3. Initialization of JSON Output:
* - A dictionary named `json\_output` is initialized with predefined values for 'age', 'gender', 'education', and 'income'.
* #### 4. Function: `generate\_criteria(individual\_df)`:
* - This function takes a DataFrame (`individual\_df`) as input and generates criteria based on specific columns.
* - It updates the global `json\_output` dictionary with criteria for 'age', 'gender', 'education', 'income', and a data profiling summary.
* - Age Criteria:
  + Checks if the 'age' column exists in the DataFrame ('age' in individual\_df.columns) and if it has more than one unique value (individual\_df['age'].nunique()).
  + If both conditions are met, it calculates and updates the 'age' criteria in the global json\_output dictionary with the 25th and 75th percentiles.
* Gender Criteria:
  + Checks if the 'gender' column exists and has more than one unique value.
  + If both conditions are met, it updates the 'gender' criteria in the global json\_output dictionary with the top two most frequent values.
* Education Criteria:
  + Checks if the 'education' column exists and has more than one unique value.
  + If both conditions are met, it updates the 'education' criteria in the global json\_output dictionary with the top three most frequent values.
* Income Criteria:
  + Checks if the 'income' column exists and has more than one unique value.
  + If both conditions are met, it updates the 'income' criteria in the global json\_output dictionary with the top five most frequent values.
* #### 5. Main Code:
* - The main code block is enclosed in a try-except block to handle exceptions gracefully.
* - \*\*Command-line Argument Parsing:\*\*
* - Using `argparse`, a command-line argument (`--file\_path` or `-f`) is expected to provide the path of a CSV file with a specific naming convention (ending with "-preprocessv2.csv").
* - \*\*Reading Data:\*\*
* - The CSV file specified in the command-line argument is read into a Pandas DataFrame (`individual\_df`).
* - \*\*Data Selection:\*\*
* - Specific columns ('gender', 'education', 'householdincome', 'age') are selected from the DataFrame.
* - \*\*Criteria Generation:\*\*
* - The `generate\_criteria` function is called with the selected DataFrame to generate criteria for the population.
* - The criteria include information about age, gender, education, income, and data profiling summary.
* - \*\*Logging and Output File Creation:\*\*
* - Logging messages are generated at various steps.
* - The criteria JSON is written to an output file in JSON format.
* - If the criteria include a 'profile\_output' key, it is removed before logging to avoid clutter.
* - \*\*Logging Success or Failure:\*\*
* - If the code execution is successful, a success message is logged with details such as the original file path and criteria.
* - If an exception occurs, an error message is logged with details about the failure and the exception.
* #### 6. Exception Handling:
* - Exceptions are caught, and the error details, including the original file path and criteria at the time of failure, are logged.
* #### 7. Overall Purpose:
* - The code is designed to read a CSV file, perform specific operations on the data, generate criteria based on certain columns, and then output the criteria and data profiling summary in a JSON file. Additionally, it provides logging to track the execution flow and handle errors gracefully.
* #### 8. Note:
* - The provided code assumes that the input CSV file follows a specific naming convention ("-preprocessv2.csv") and contains columns such as 'gender', 'education', 'householdincome', and 'age'. Adjustments may be needed based on actual data and requirements.