**Low Level Design(LLD)**

Prediction of LC50 value using Quantitative structure–activity

relationship models (QSAR models)

|  |  |
| --- | --- |
| Done by | Hariprasath G |
| Document Version | 0.2 |
| Last Revised Date | 07-07-2023 |

**Document Control**

**Change Record:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
| 0.1 | 01-07-2023 | Hariprasath G | Introduction and System Architecture |
| 0.2 | 05-07-2023 | Hariprasath G | Architecture Description and Test Report |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Reviews:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Reviewer** | **Comments** |
|  |  |  |  |

**Approval Status:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Review Date** | **Reviewed by** | **Approved by** | **Commnets** |
|  |  |  |  |  |

|  |  |
| --- | --- |
| **Contents** |  |
| 1. Introduction……………………………………………….. | 4 |
| 1.1 What is Low-Level Design Document?......................... | 4 |
| 1.2 Scope…………………………………………………. | 4 |
| 1. System Architecture………………………………………. | 5 |
| 1. Architecture Description………………………………….. | 5 |
| 3.1 Data Description……………………………………… | 6 |
| 3.2 Data Validation……………………………………….. | 6 |
| 3.3 Data Transformation…………………………………. | 7 |
| 3.3.1 Handling Duplicates……………………………. | 7 |
| 3.3.2 Handling Duplicates Level2……………………. | 8 |
| 3.3.3 Feature Validation……………………………… | 9 |
| 3.3.4 Handling Missing Values………………………. | 9 |
| 3.3.5 Handling Outliers………………………………. | 10 |
| 3.3.6 Dimensionality Reduction……………………… | 11 |
| 3.3.7 Train/Test Split…………………………………. | 12 |
| 3.4 Model Trainer………………………………………… | 12 |
| 3.4.1 Import Splitted Data……………………………. | 12 |
| 3.4.2 Trainer………………………………………….. | 13 |
| 3.4.3 Range Calculator……………………………….. | 14 |
| 3.4.4 Report Generator……………………………….. | 15 |
| 3.5 Model Evaluation…………………………………….. | 16 |
| 3.5.1 Evaluate………………………………………… | 16 |
| 3.6 Utlility………………………………………………... | 17 |
| 3.6.1 Create Dictionary………………………………. | 17 |
| 3.6.2 Models………………………………………….. | 17 |
| 3.6.3 Remove Unwanted Columns…………………… | 18 |
| 3.6.4 Column Names…………………………………. | 18 |
| 3.6.5 Import Splitted Data……………………………. | 19 |
| 3.6.6 Save and Load………………………………….. | 19 |
| 3.6.7 Import Custom Splitted Data…………………... | 20 |
| 3.6.8 Custom Model Training………………………... | 21 |
| 3.6.9 Filtered Report…………………………………. | 21 |
| 3.7 Model Loader…………………………………………. | 22 |
| 3.8 Application……………………………………………. | 23 |
| 1. Test Report………………………………………………... | 24 |
|  |  |

1. **Introduction**
   1. **What is Low-Level Design Document?**

Low-level design refers to the detailed and specific design specifications for individual components or modules within a system. It involves defining the internal workings, interfaces, algorithms, data structures, error handling, security measures, performance optimizations, and other implementation details required for the development and integration of the system components.

* 1. **Scope**

The scope of a low-level design document, encompasses the detailed design specifications and considerations for individual components or modules within a system. It involves defining the internal workings, interfaces, and interactions of each component, as well as addressing specific design aspects like data structures, algorithms, error handling, security measures, and performance optimization. The document outlines the granular details of the system's design, including the model architecture, data preprocessing steps, feature engineering techniques, hyperparameter tuning, model training and evaluation procedures, deployment architecture, monitoring mechanisms, and maintenance guidelines. It serves as a comprehensive guide for developers to implement and integrate the components effectively, ensuring a cohesive and well-designed system that meets the project's requirements.

1. **System Architecture**

****



Start

Data Validation

Handling Duplicates

Handling Duplicates level 2

Feature Validation

Handling Missing Values

Handling Outliers

Dimensionality Reduction

Import Data

Generate Report

Import Splitted Data

Model Trainer

Models

Calculate Error Range

Model Evaluation

Underfitted /Overfitted

Filtered Report

Custom Splitted Data

Custom Model Training



Model

Yes

No

Optimal Model

Model Loader

Front-End

Data From User

Prediction

End

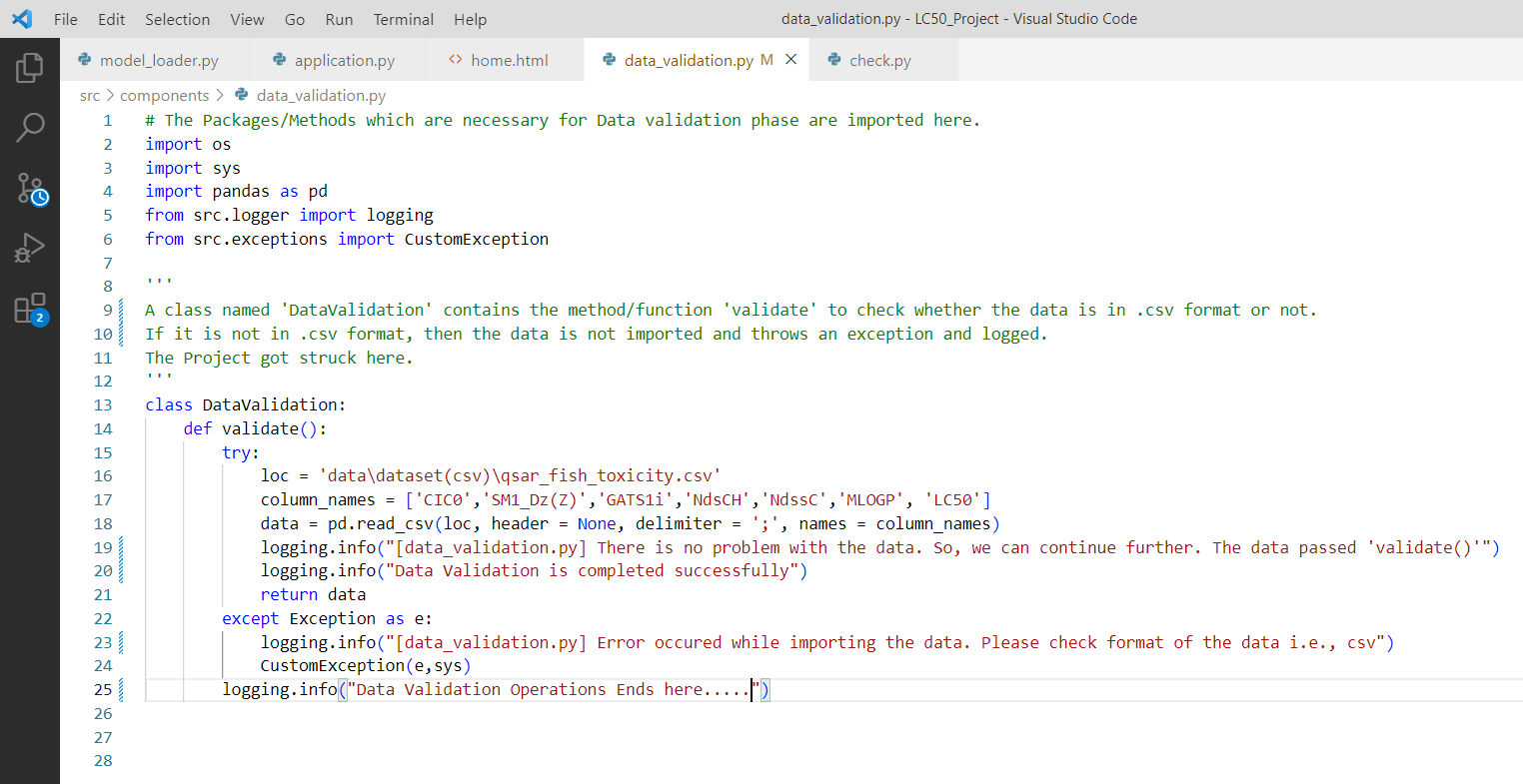
Application

1. **Architecture Description**
   1. **Data Description**

Data set containing values for 6 attributes (molecular descriptors) of 908 chemicals used to predict quantitative acute aquatic toxicity towards the fish Pimephales promelas (fathead minnow).

This dataset was used to develop quantitative regression QSAR models to predict acute aquatic toxicity towards the fish Pimephales promelas (fathead minnow) on a set of 908 chemicals. LC50 data, which is the concentration that causes death in 50% of test fish over a test duration of 96 hours, was used as model response. The model comprised 6 molecular descriptors: MLOGP (molecular properties), CIC0 (information indices), GATS1i (2D autocorrelations), NdssC (atom-type counts), NdsCH ((atom-type counts), SM1\_Dz(Z) (2D matrix-based descriptors). Details can be found in the quoted reference: M. Cassotti, D. Ballabio, R. Todeschini, V. Consonni. A similarity-based QSAR model for predicting acute toxicity towards the fathead minnow (Pimephales promelas), SAR and QSAR in Environmental Research (2015), 26, 217-243; doi: 10.1080/1062936X.2015.1018938

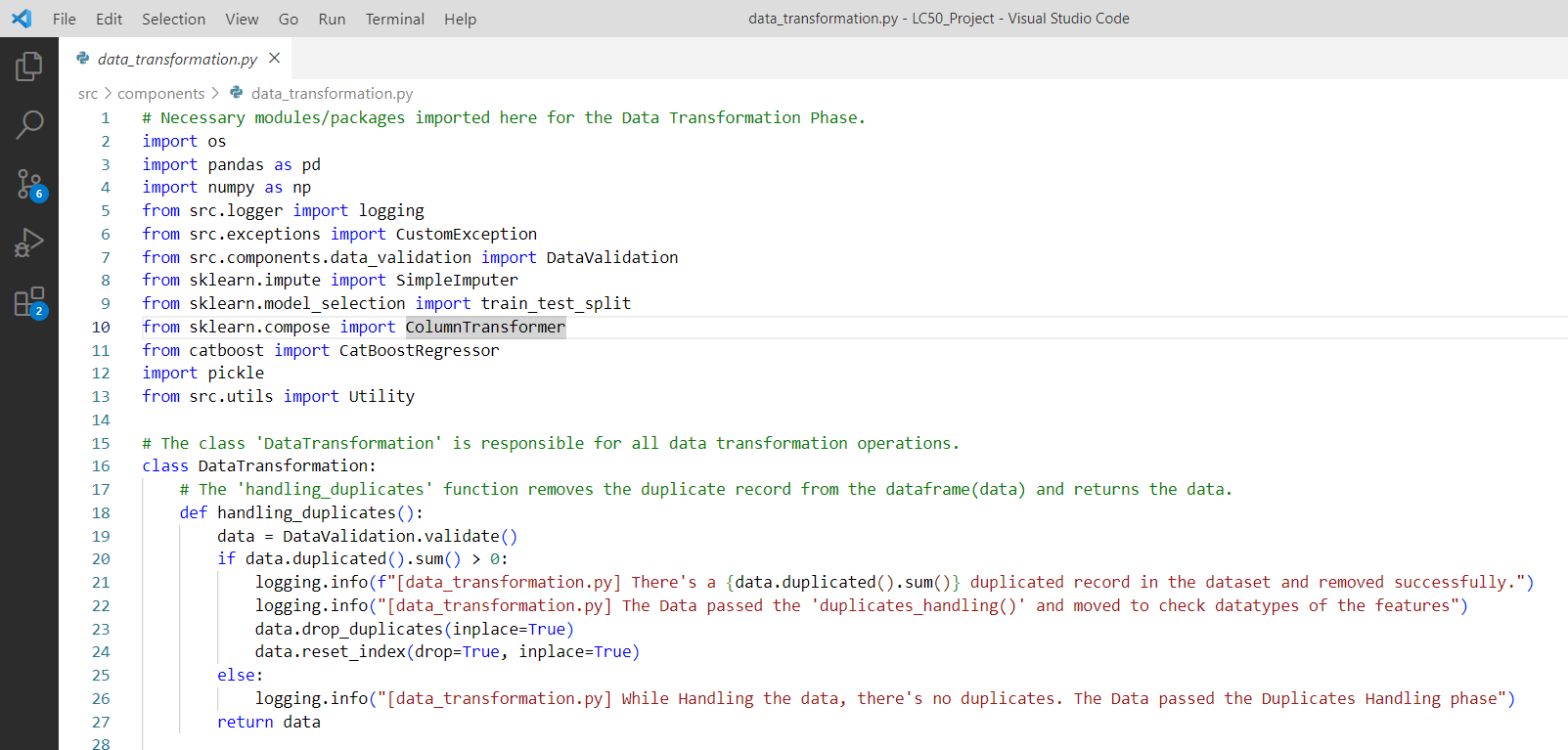
* 1. **Data Validation**

****

The class ‘DataValidation’ from ‘data\_validation.py’ file of folder located in ‘src/components/’ is resposible for all operations of data validation. The Necessary packages like logger, exceptions, os and pandas were imported. The class ‘DataValidation’ contains only one function i.e., validate(). In try block, the desired location/path of the dataset is assigned to ‘loc’, list of features names were assigned to ‘column\_names’ and the data is read with the help of ‘read\_csv’ method of ‘pandas’ package, then assigned to ‘data’. The message inside the logging function is recorded only if all operations mentioned above in the try block were successfully completed. Or else the except block is triggered and raises an exception in the terminal and log will be recorded as ‘Error Occured’.

* 1. **Data Transformation**

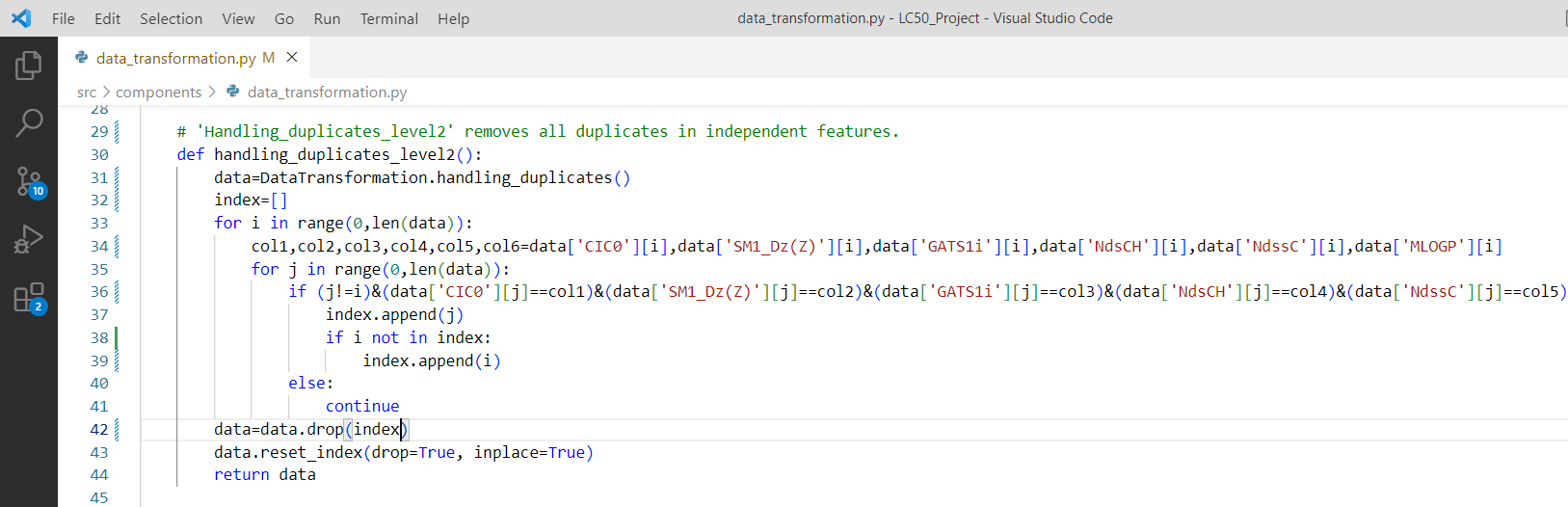
**3.3.1 Handling Duplicates**

****

The class ‘DataTransformation’ from data\_transformation.py file of folder ‘src/components/’ is responsible for all operations related to data transformation like handling duplicates, checking data type of the features, handling missing values, handling outliers, dimensionality reduction and splitting dataset. The necessary packages like sklearn, pickle, uitls, etc., were imported.

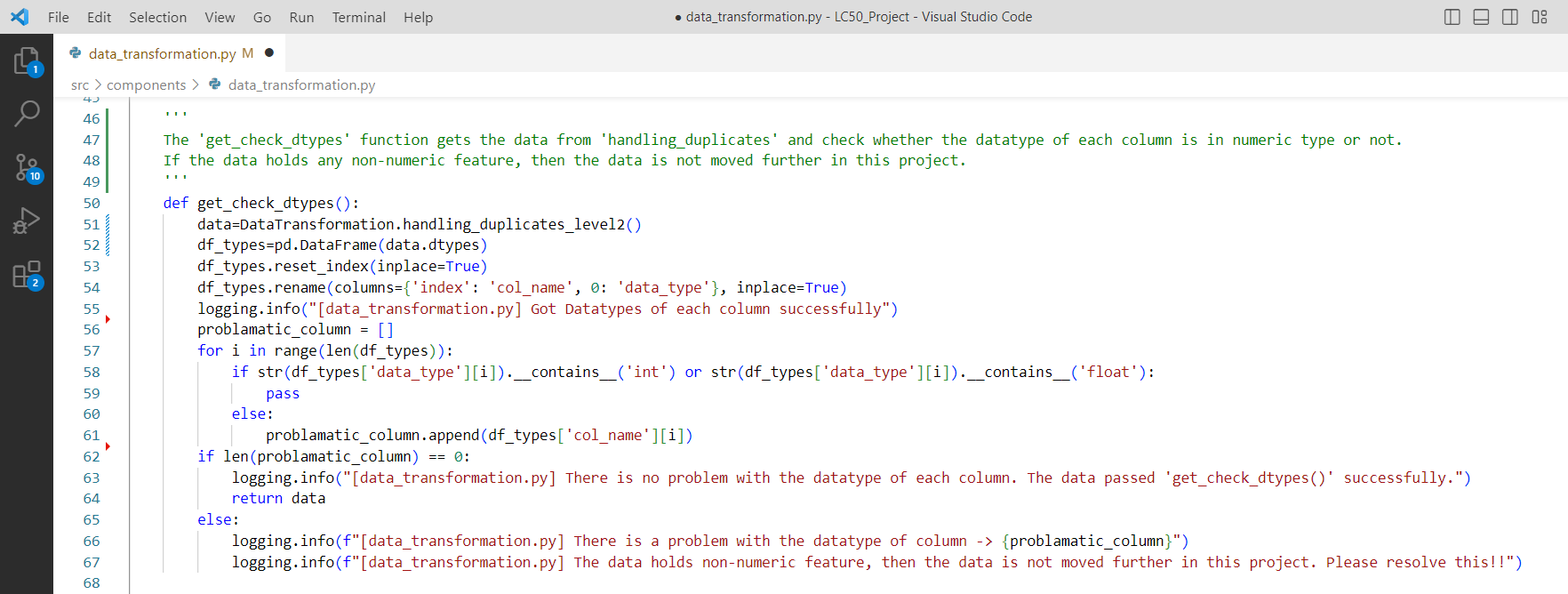
The dataset returned from ‘validate’ function of ‘data\_validation.py’ is called here and assigned to the variable ‘data’. The ‘if block’ of function ‘handling\_duplicates’ checks for the duplicate counts in the dataset and removes if the count is greater than zero with the help of ‘drop\_duplicates’ method in pandas DataFrame. The message is logged after removal of duplicates from the dataset. If no duplicates found ‘else’ part is triggered and the message is logged successfully. Then, returns the data.

**3.3.2 Handling Duplicates Level2**

****

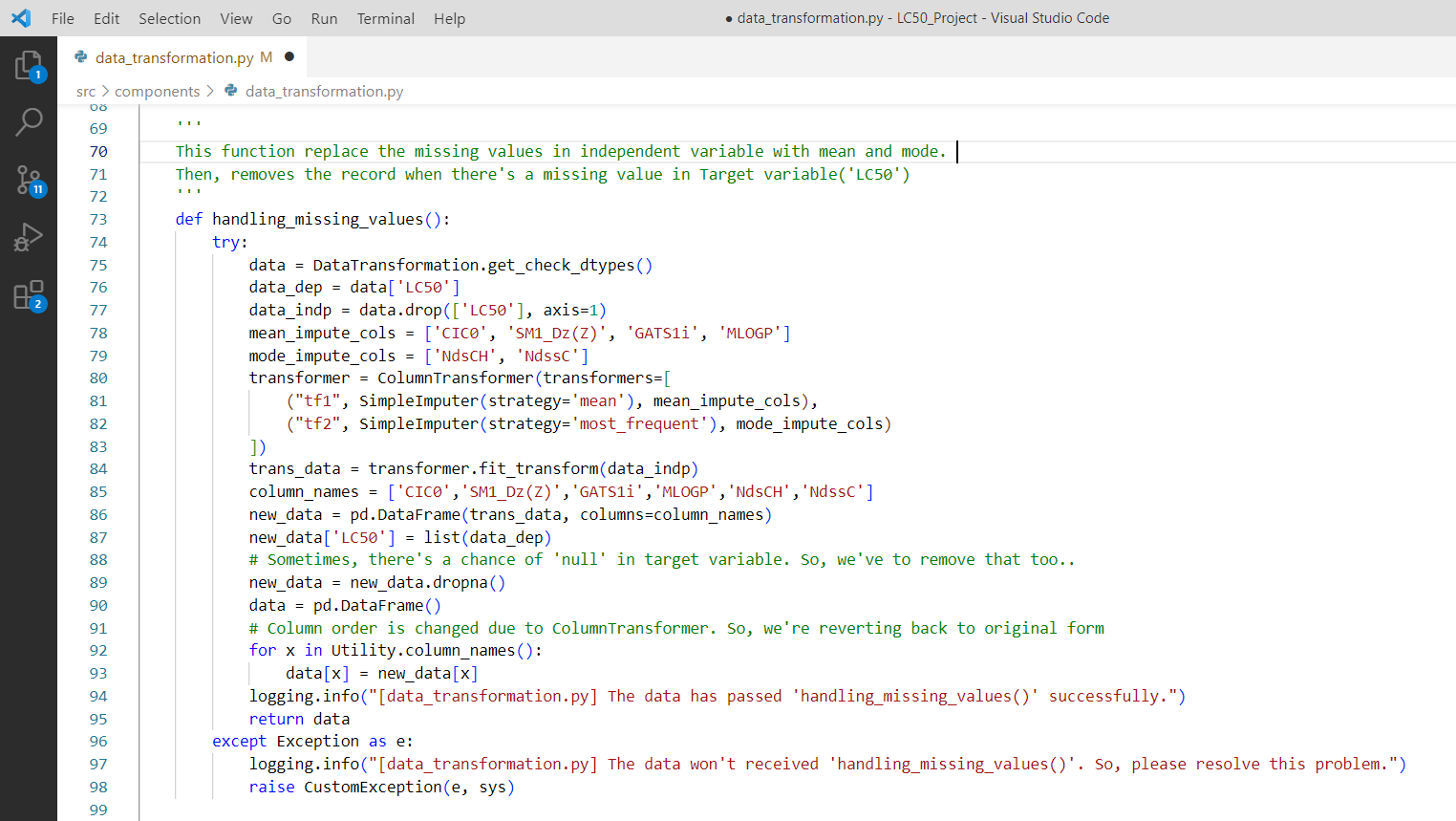
The ‘handling\_duplicates\_level2’ function removes all records of duplicates in independent feature. The data returned from the function ‘handling\_duplicates’ of ‘data\_transformation.py’ is called here and assigned to the variable ‘data’. Here, we’ve created an algorithm to remove all duplicated independent records as well as original record because we’re not sure which record is original. ‘for’ loop run for number of times which is equal to the length of data. Every single record is compared with all other record without considering target variable and index value of current record as well as the index of found record were updated in the list(index). The index values in the dataset is reseted using ‘reset\_index’ method of pandas. Finally, return the data.

**3.3.3 Feature Validation**



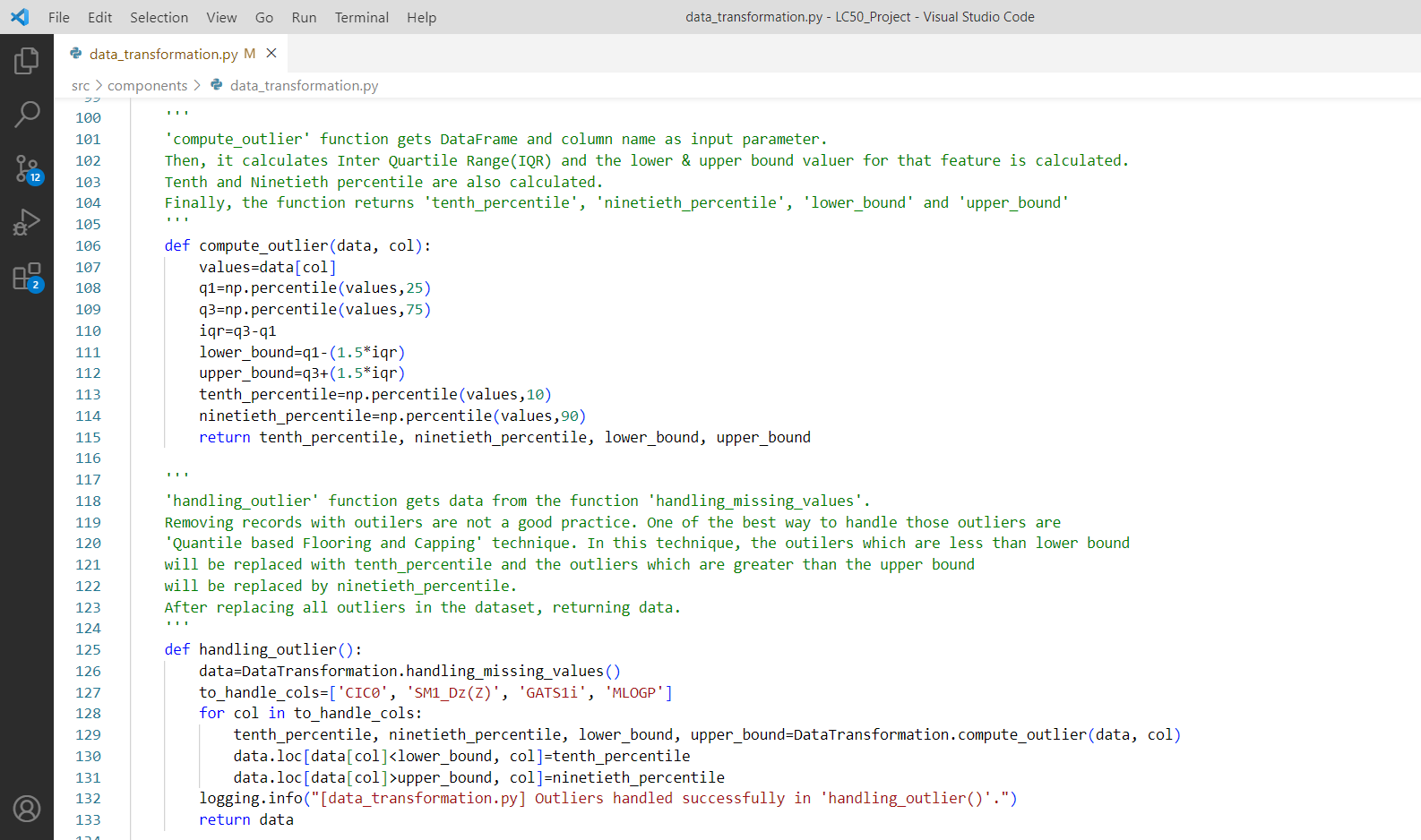
The ‘get\_check\_dtypes’ function checks the datatype of all features in dataset. The data returned from the function ‘handling\_duplicates\_level2’ of ‘data\_transformation.py’ is called here and assigned to the variable ‘data’. The datatypes of data is recorded in DataFrame and First column is renamed as ‘col\_name’ and Second column as ‘data\_type’. We’ve created a logic to check the datatypes of all features in the data of int/float. If no record found with other datatypes, the data is returned and waits for function call. Then, the message of operation completion is logged using ‘logging’ function.

**3.3.4 Handling Missing Values**

****

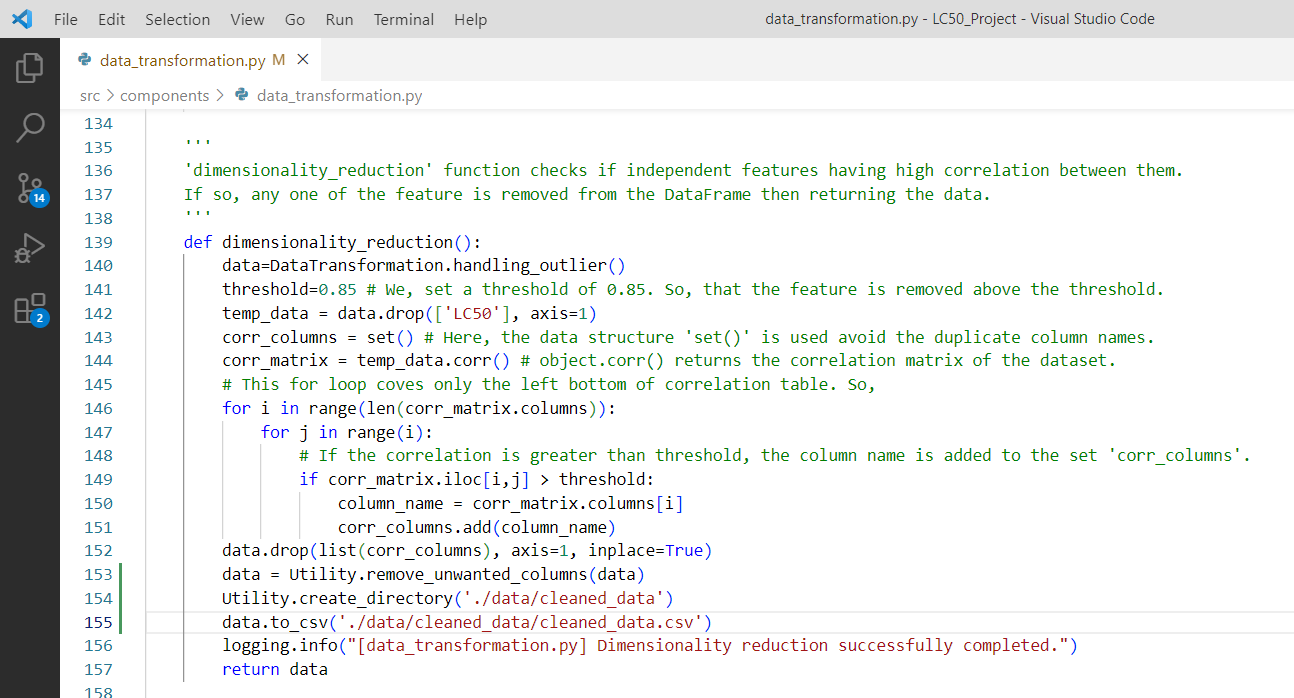
The ‘handling\_missing\_values’ function replaces all missing values with the help of ‘ColumnTransformer’ and ‘SimpleImputer’ in sklearn package. In try block, the data returned from the ‘get\_check\_dtypes’ function is called and assigned to the variable ‘data’. There are 4 continuous feature and 2 categorical feature in the independent variable. The missing values of the features with continuous data are replaced with mean and the missing values of features with categorical data are replaced with mode(most\_frequent). Then the dataset is arranged in the original form and returned. If error occurs in the try block, then ‘except’ block is trigger and raises an exception and the messages were logged.

**3.3.5 Handling Outliers**



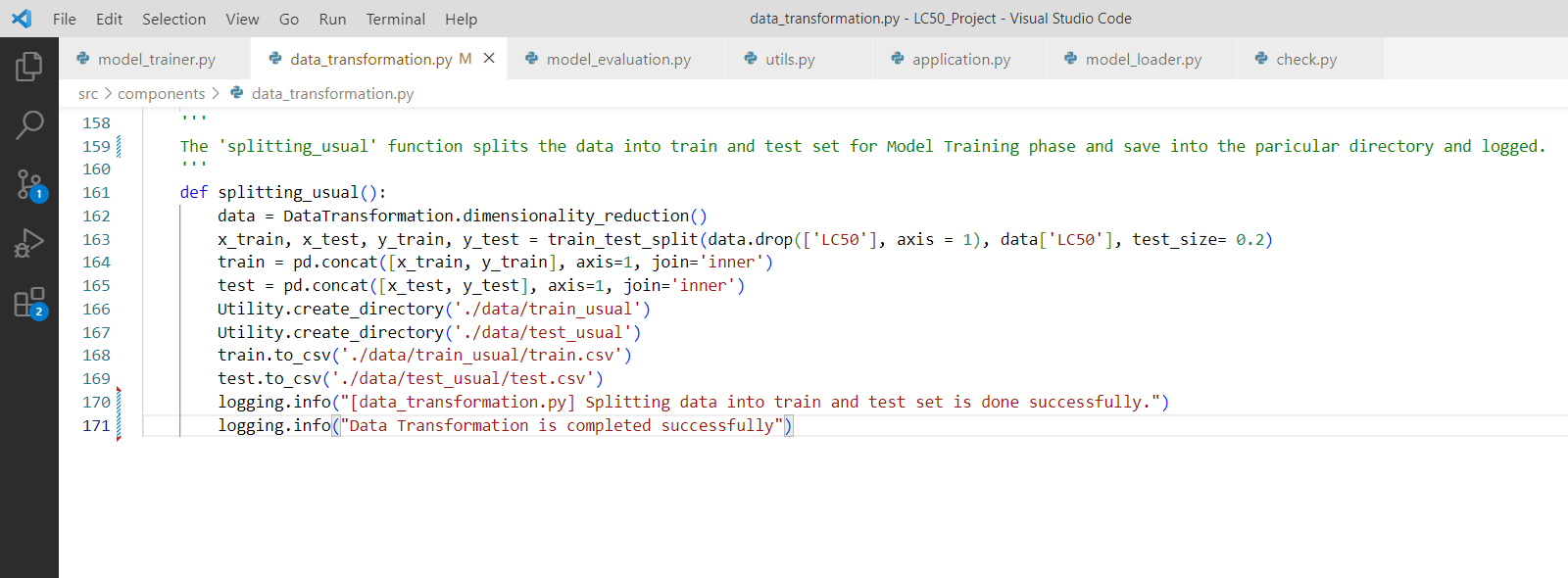
The ‘handling\_outlier’ function replaces all outliers in the features of continuous data. The data returned from ‘handling\_missing\_values’ function are called and assigned to the variable ‘data’. The continuous features were defined in the variable ‘to\_handle\_cols’. The ‘for’ loop iterates through the variable ‘to\_handle\_cols’. The ‘compute\_outlier’ function takes input parameter as data and column name of ‘to\_handle\_cols’ in for loop. So that the function ‘compute\_outlier’ calculates the 25th percentile, 75th percentile, Inter Quantile Range, lower bound, upper bound, 10th percentile and 90th percentile. Then, returns 10th percentile, 90th percentile, lower bound and upper bound of the iput column. Then the datapoint which is greater than the upper bound value is replaced with 90th percentile and the datapoint which is lesser than the lower bound value is replaced with 10th percentile. Then, operation completion message is logged. Finally, return the data.

**3.3.6 Dimensionality Reduction**

****

The ‘dimensionality\_reduction’ function removes any one of the feature having high correlation with respect to other feature. The data returned from ‘handling\_outlier’ function is called and assigned to the variable ‘data’. We have to set a threshold manually with the float value nearer to 1.0(maximum value). Here, we set a threshold of 0.85 manually. So that, any one of the feature having high correlation with other feature is removed. The data is saved in desired folder as ‘cleaned\_data’ because the data preprocessing phase ends here. Then, operation completion message will be logged. Finally, return the data.

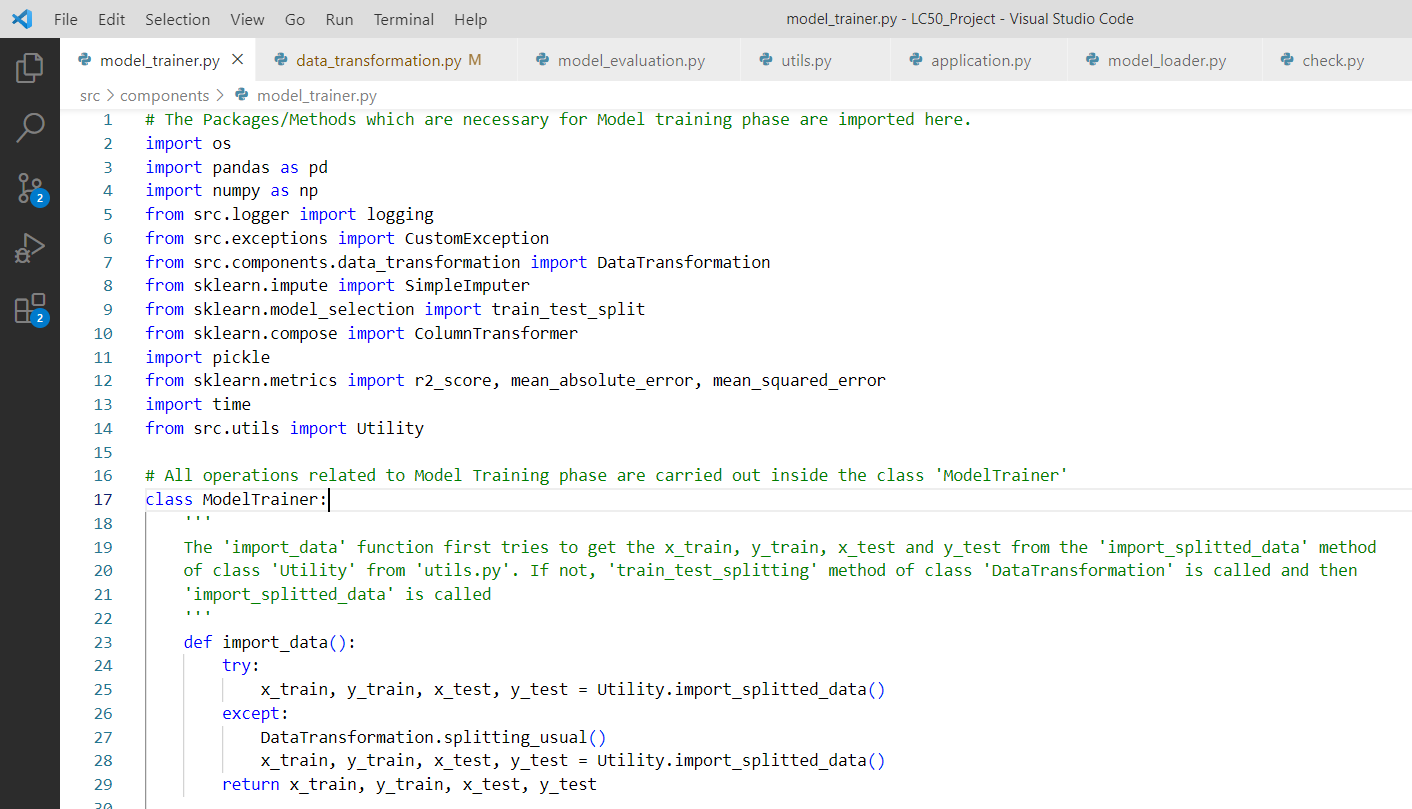
**3.3.7 Train/Test Split**

****

The ‘splitting\_usual’ function splits the data into train and test set for model preparation. The data returned from ‘dimensionality\_reduction’ is called and assigned to the variable ‘data’. From the ‘model\_selection’ of sklearn package ‘train\_test\_split’ method/function is used. Independent and Dependent variable are passed as an input parameter and ‘test\_size’ of 20% were defined. These train and test were saved in the particular directory in .csv(Comma Separated Value) format and the message of operation completion is logged.

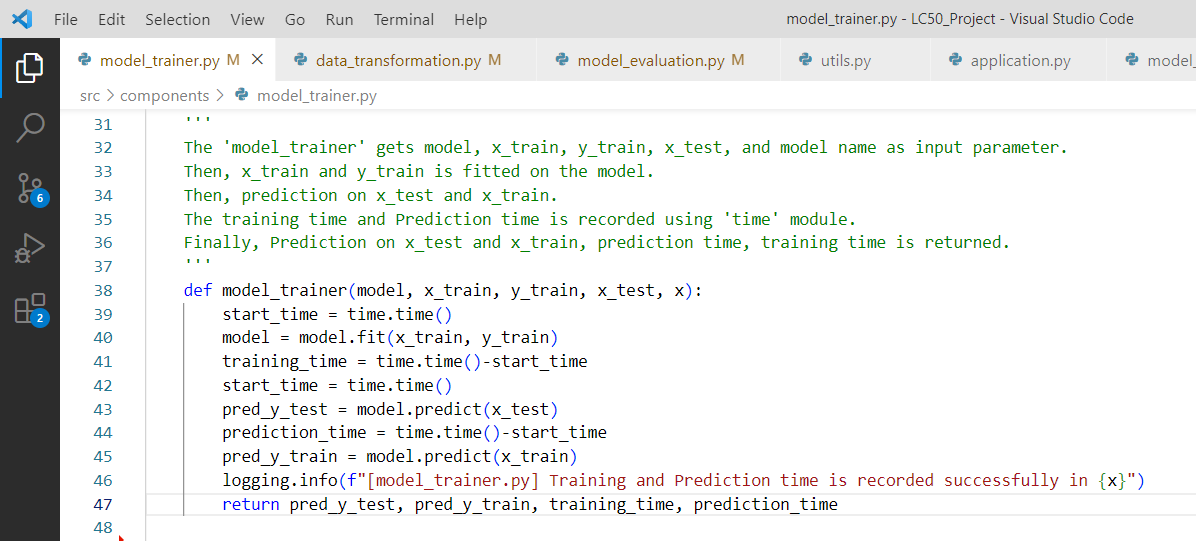
**3.4 Model Trainer**

**3.4.1 Import Splitted Data**

****

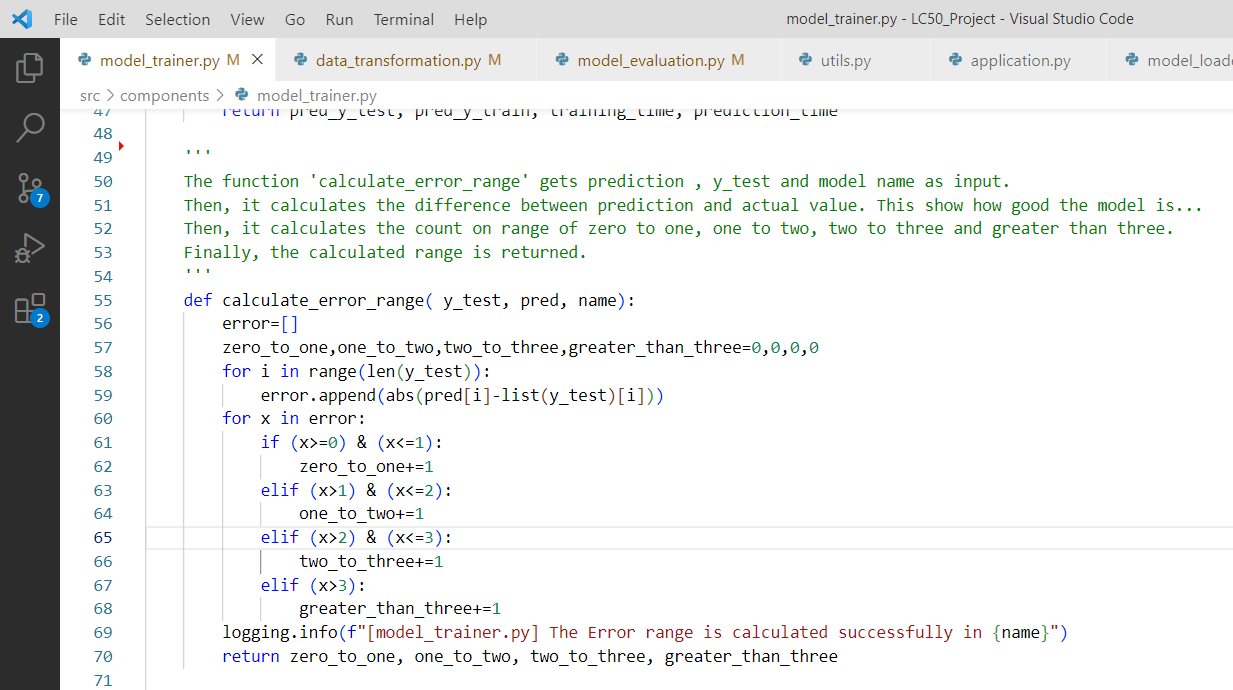
The Necessary packages like numpy, pandas, sklearn, etc., were imported for the Model Training phase. In this module, the class ‘ModelTrainer’ is responsible for all operations related to model training. The first function defined in the class is ‘import\_data’. In try block, the function calls the ‘import\_splitted\_data’ function from ‘Utility’ class of ‘utils’. The ‘import\_splitted\_data’ gets the train and test file from the location defined in ‘splitting\_usual’ function of class ‘DataTransformation’ and assigned to the variable ‘x\_train’, ‘y\_train’, ‘x\_test’ and ‘y\_test’. Else, ‘except’ block triggered ‘splitting\_usual’ function is called to perform train and test split. Finally, return ‘x\_train’, ‘y\_train’, ‘x\_test’ and ‘y\_test’.

**3.4.2 Trainer**

****

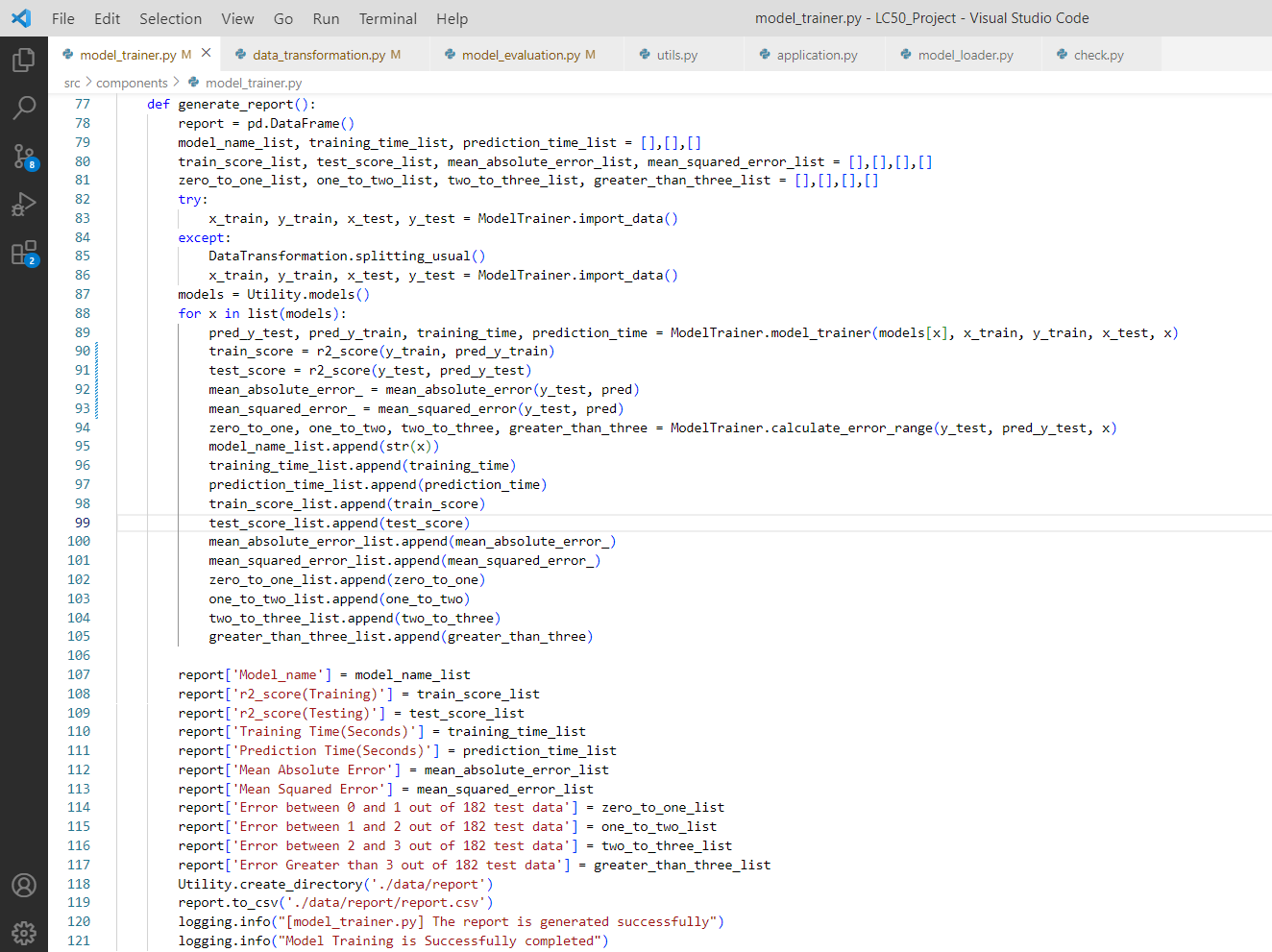
The function ‘model\_trainer’ is responsible for producing an output of training and prediction time of the model, prediction on ‘x\_test’ and prediction on ‘x\_train’. The function takes ‘model’, ‘x\_train’, ‘x\_test’, ‘y\_train’, ‘y\_test’ and Algorithm name as input parameter. Time is recorded for model fitting and model prediction on ‘x\_test’ using ‘time’ module. Then, prediction on ‘x\_test’ and ‘x\_train’ is done and returned.

**3.4.3 Calculate Error Range**

****

The ‘calculate\_error\_range’ function returns the count of records haviong error range of 0 to 1, 1 to 2, 2 to 3 and greater than 3. The benefit of this function is get an idea about the performance of the model. The function takes input as ‘y\_test’ , ‘predictions’ and ‘algorithm name’. Then, absolute difference of prediction and true value is calculated and stored in a list(error). Each value in the list is checked and counted. Finally, returns the count of 0 to 1, 1 to 2, 2 to 3 and greater than 3.

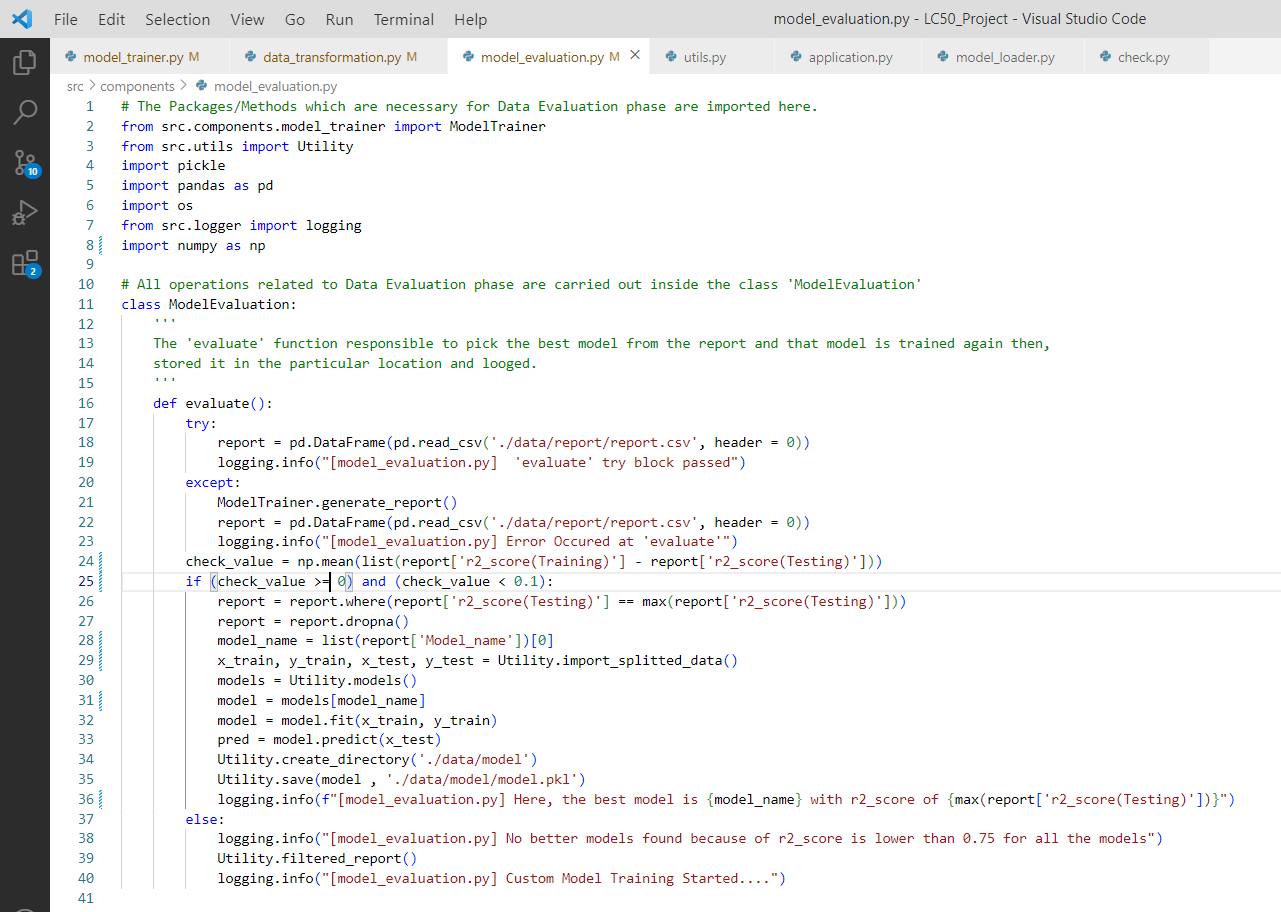
**3.4.4 Generate Report**

****

The function ‘generate\_report’ generates a report of information includes Name of the Algorithm, R2 Score of Training and Testing set, Training Time, Prediction Time, Mean Absolute Error, Mean Squared Error, Error ranges from ‘calculate\_error\_range’ i.e., 0 to 1, 1 to 2, 2 to 3 and Greater than 3. An empty dataframe of name ‘report’ is defined. ‘x\_train’, ‘y\_train’, ‘x\_test’ and ‘y\_test’ were imported by calling ‘import\_data’ function. A dictionary of regression algorithms were called from ‘models’ function from ‘utils’ and assigned to the variable ‘models’. Each model is iterated over the for loop. In the for loop, the function ‘model\_trainer’ is called with input parameters of Regression Algorithm, x\_train, y-train, x\_test and name of the algorithm and assigned to desired variables. The R2 score(both train and test score), mean absolute error and mean squared error of testing set is calculated and assigned to desired variable. Then, ‘calculate\_error\_range’ function is called and error range counts were assigned to desired variables. The data stored in each variable is added to the desired list. Finally, those lists were assigned to the column of ‘report’ dataframe and saved the report in .csv format on a desired location and logged successfully.

**3.5 Model Evaluation**

**3.5.1 Evaluate**

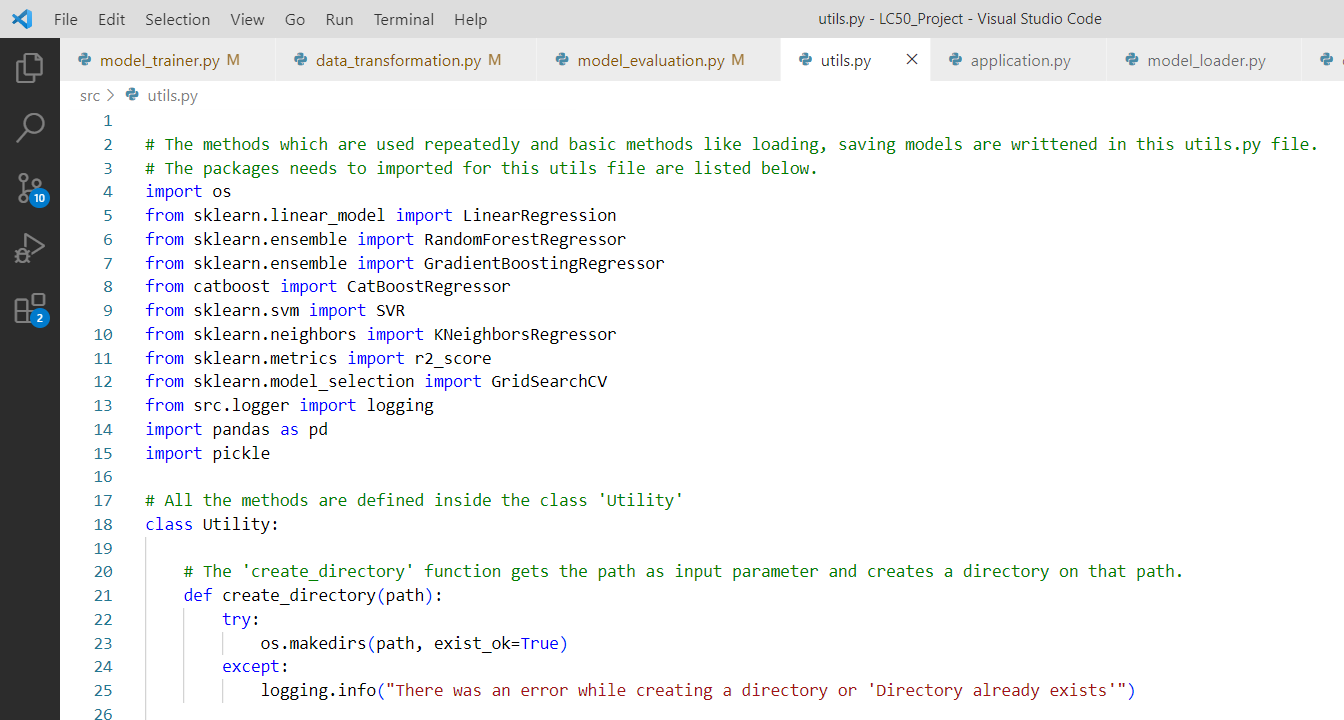
****

The module ‘model\_evaluation’ take cares of all operations related to model evaluation. The function ‘evaluate’ in the class ‘ModelEvaluation’ tries to get the read the report file which were geenerated from the ‘generate\_report’ function of ‘model\_trainer’ module. The mean of difference between the training and testing score for all algorithm is calculated. The calculated mean value should be less than zero or greater than 0.1, otherwise the report is not considered. The reason is, ‘The possibility of underfitting or overfitting occurred in the data’. Both of the scenario should be avoided. Then, moved to get a generalized model.

**3.6 Utility**

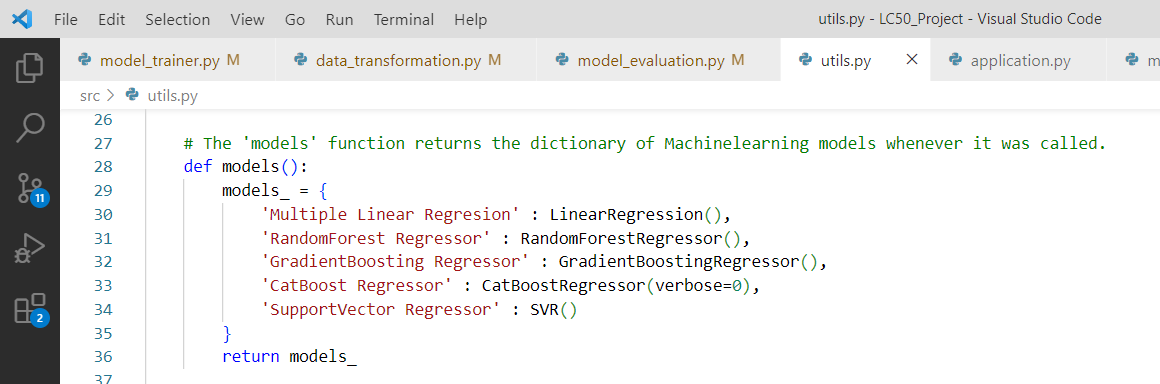
The module ‘utils’ contains a class ‘Utility’. The functions defined in the class ‘Utility’ are reusable functions from different modules which includes ‘create\_directory’, ‘models’, ‘remove\_unwanted\_columns’, ‘import\_splitted\_data’, ‘load’ and ‘save’. Few functions in the class ‘Utility’ were called only once in the application are, ‘custom\_model\_training’, ‘filtered\_report’ and ‘import\_custom\_splitted\_data’.

**3.6.1 Create Directory**

****

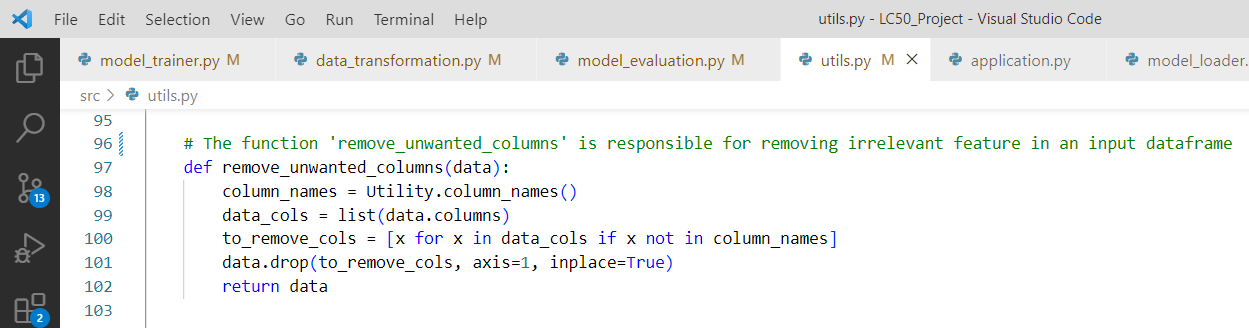
The ‘create\_directory’ function needs one input parameter i.e., the path or location where we need to make a directory. In try block, ‘makedirs’ method of ‘os’ module is used to create a directory on desired path/location. Else, the message of error occurance is logged.

**3.6.2 Models**

****

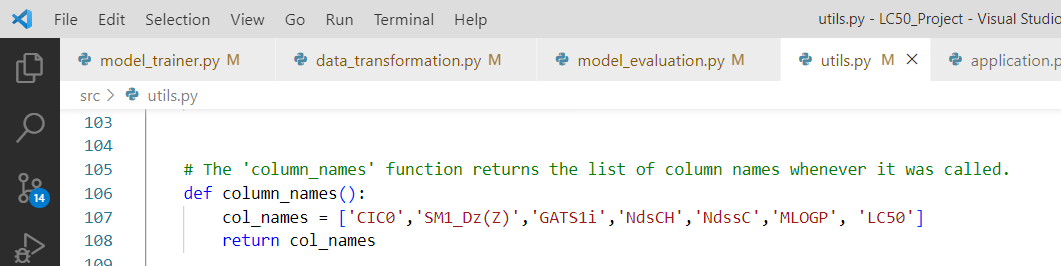
The function ‘models’ is responsible for returning all Machine Learning Algorithms which were stored in a dictionary whenever called. A Dictionary of Machine Learning Algorithms for Regression is assigned for the keys of algorithm names and is assigned to the variable ‘models\_’. Finally, returns the ‘models\_’.

**3.6.3 Remove Unwanted Columns**

****

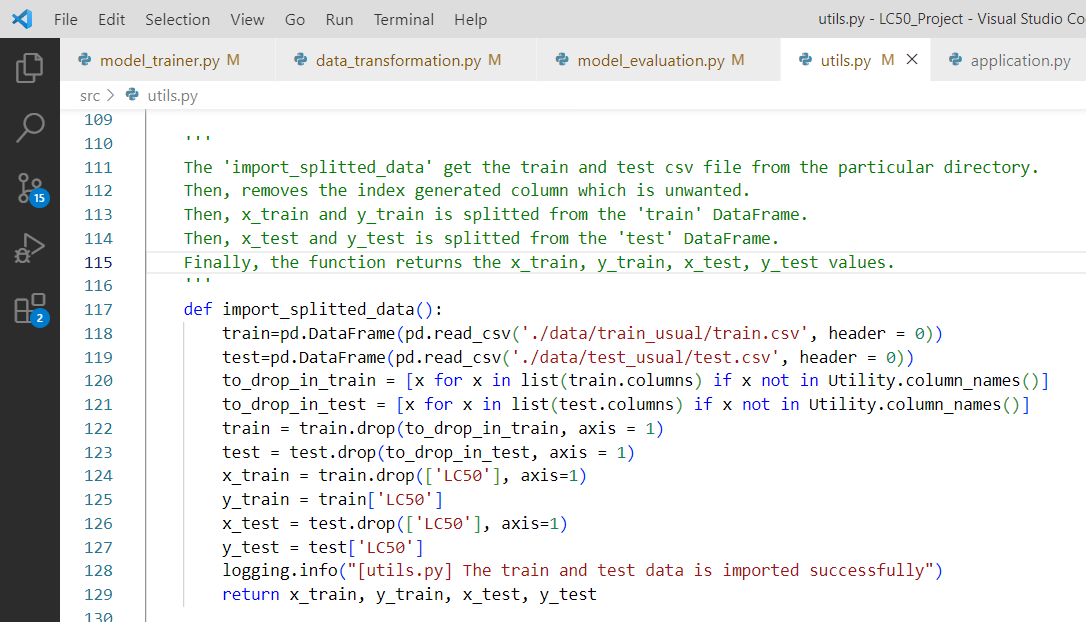
The function ‘remove\_unwanted\_columns’ is responsible for removing all irrelevant features in the input data. The feature names of the project were stored in a function ‘column\_names’ of ‘utils’ were assigned to the variable ‘column\_names’. The feature names of the input data is compared with the project’s feature names. If any of the feature from input data is not present in project’s feature names, the feature will be removed input data and updated using ‘inplace’ property. Finally, returns the data.

**3.6.4 Column Names**

****

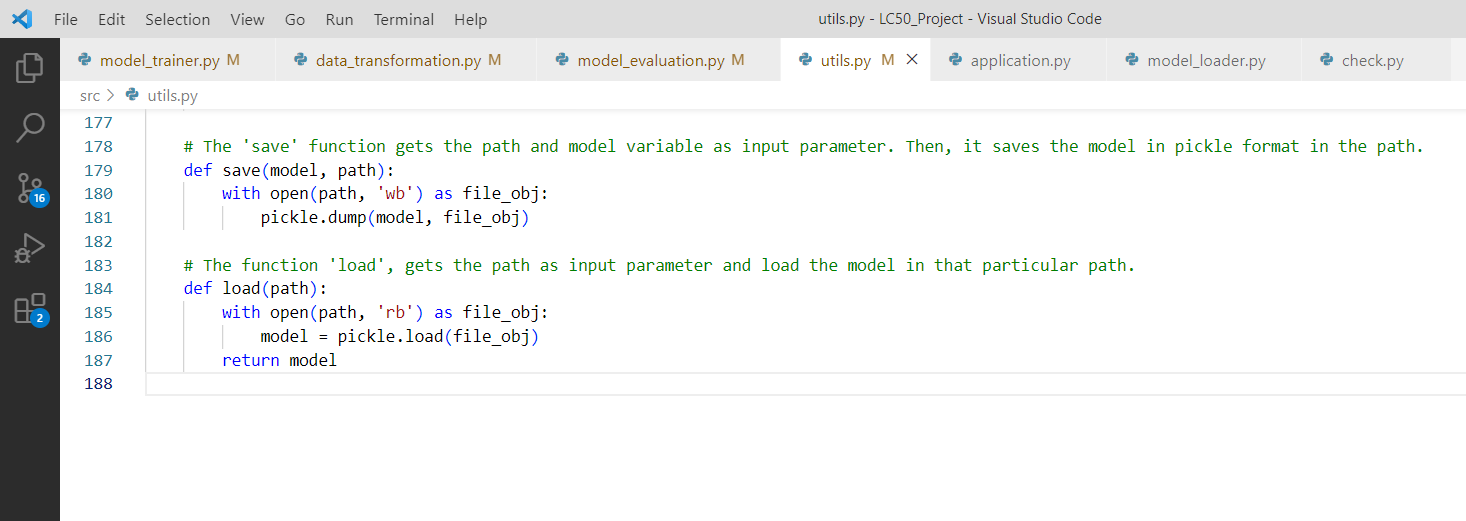
The function returns the list of feature names of the project whenever it was called.

**3.6.5 Import Splitted Data**

****

The function ‘import\_splitted\_data’ reads the train and test split file from desired location and returns ‘x\_train’, ‘y\_train’, ‘x\_test’ and ‘y\_test’ and the message were logged.

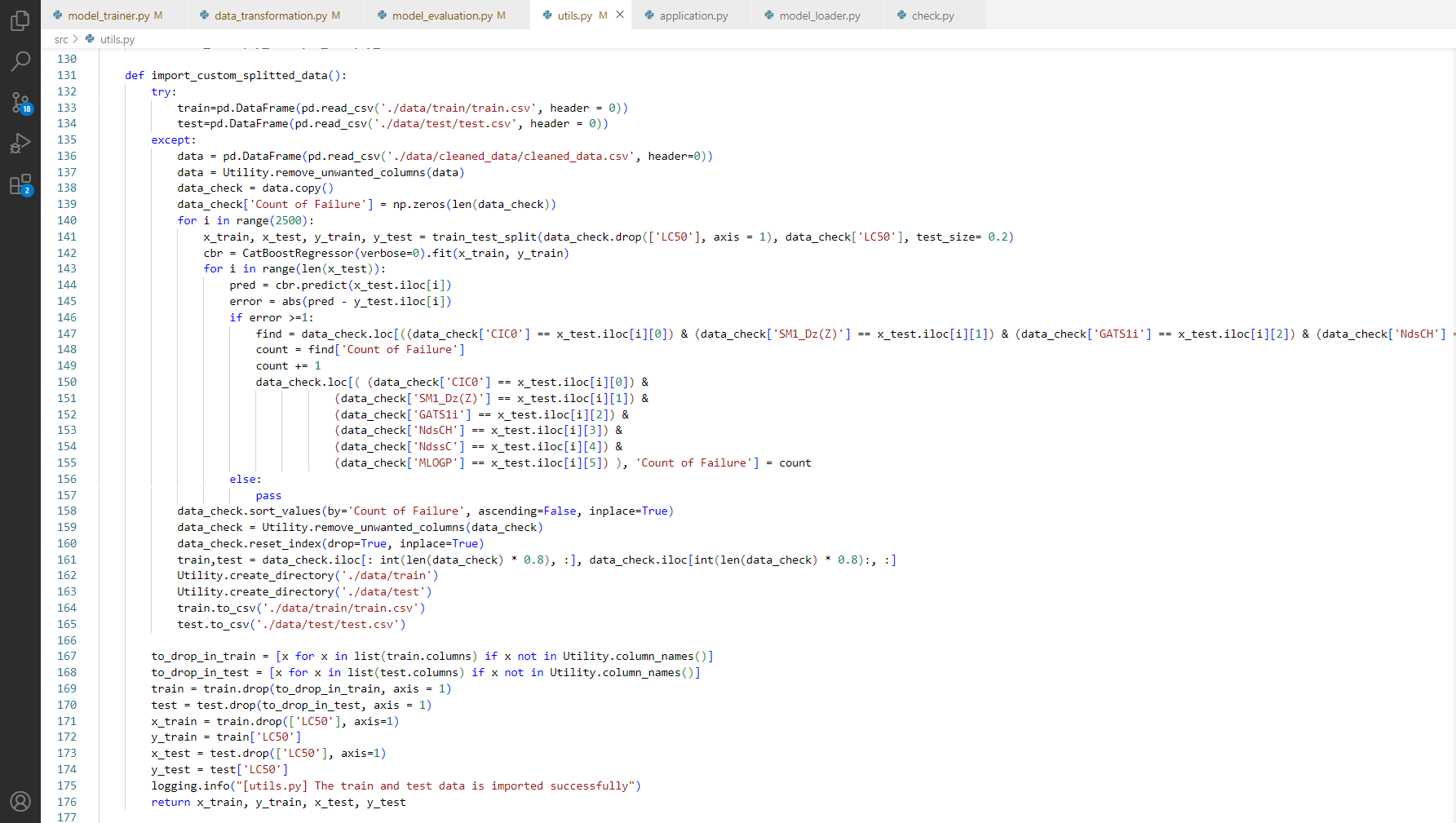
**3.6.6 Save and Load**

****

The ‘save’ function takes two input parameters of trained model and path. By using ‘pickle’ module the model is dumped in a ‘write-binary’ mode of desired location/path.

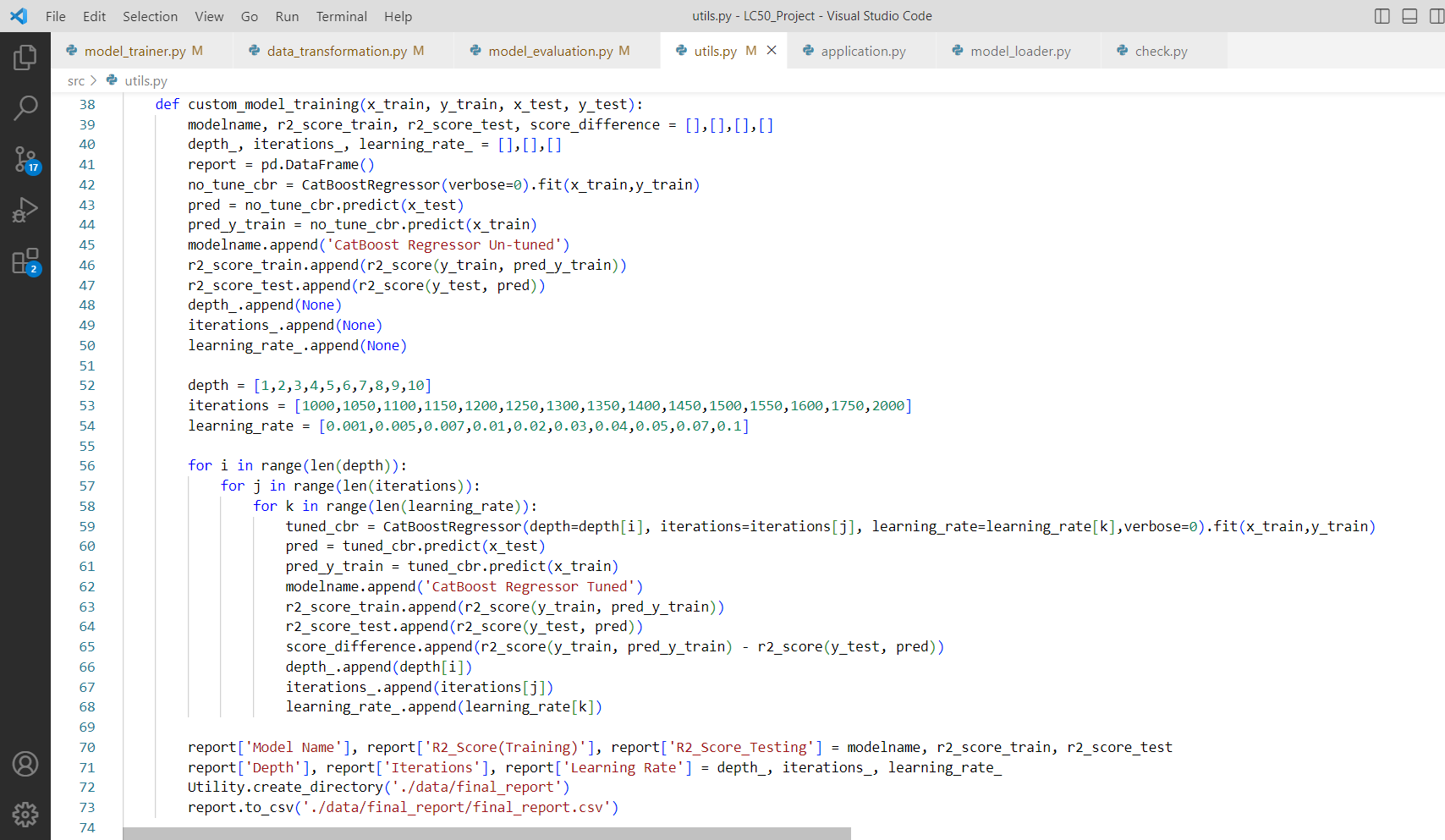
The ‘load’ function takes one input as parameter of desired location or path. It brings back the dumped pickle file in a ‘read-binary’ mode of desired location.

**3.6.7 Import Custom Splitted data**

****

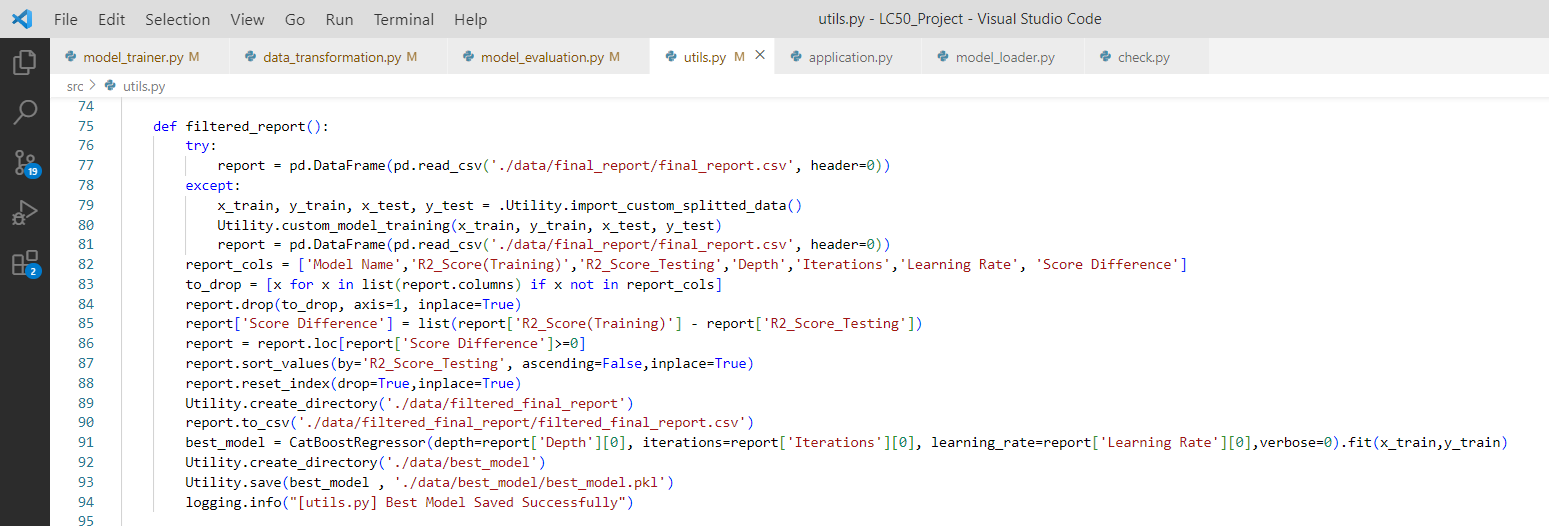
The function ‘import\_custom\_splitted\_data’ is responsible for splitting the dataset into custom train and test split only if fails in traditional way of model training with the strong reason of huge difference between train and test score. In this project, the train score of the model is around 0.95 and test score is around 0.6 or even below in some times. This is purely because of overfitting. Overfitting is nothing but the training set learns well but eventually fails when it comes to test data. It litrellay means that the records which needs to be learn was present is test data and also the dataset is non-linear in nature. So we’ve to move those records into training part to avoid this overfitting. The approach we took here is, noting down the records whose error is greater than 1.0. Here, error is nothing but the absolute difference of true and predicted value. The reason for the threshold that we fixed as 1.0 is that the error of the predictions on training set not exceeded 1.0. This process runs until 2500 iterations. The ‘count of failure’ feature tells whether the data is to be in train or test part. The dataset is arranged in a descending order of the feature ‘count\_of\_failure’. The First 80% of the data is considered as training part and the rest 20% will be moved to testing part. Finally, returns ‘x\_train’, ‘y\_train’, ‘x\_test’ and ‘y\_test’

**3.6.8 Custom Model Training**

****

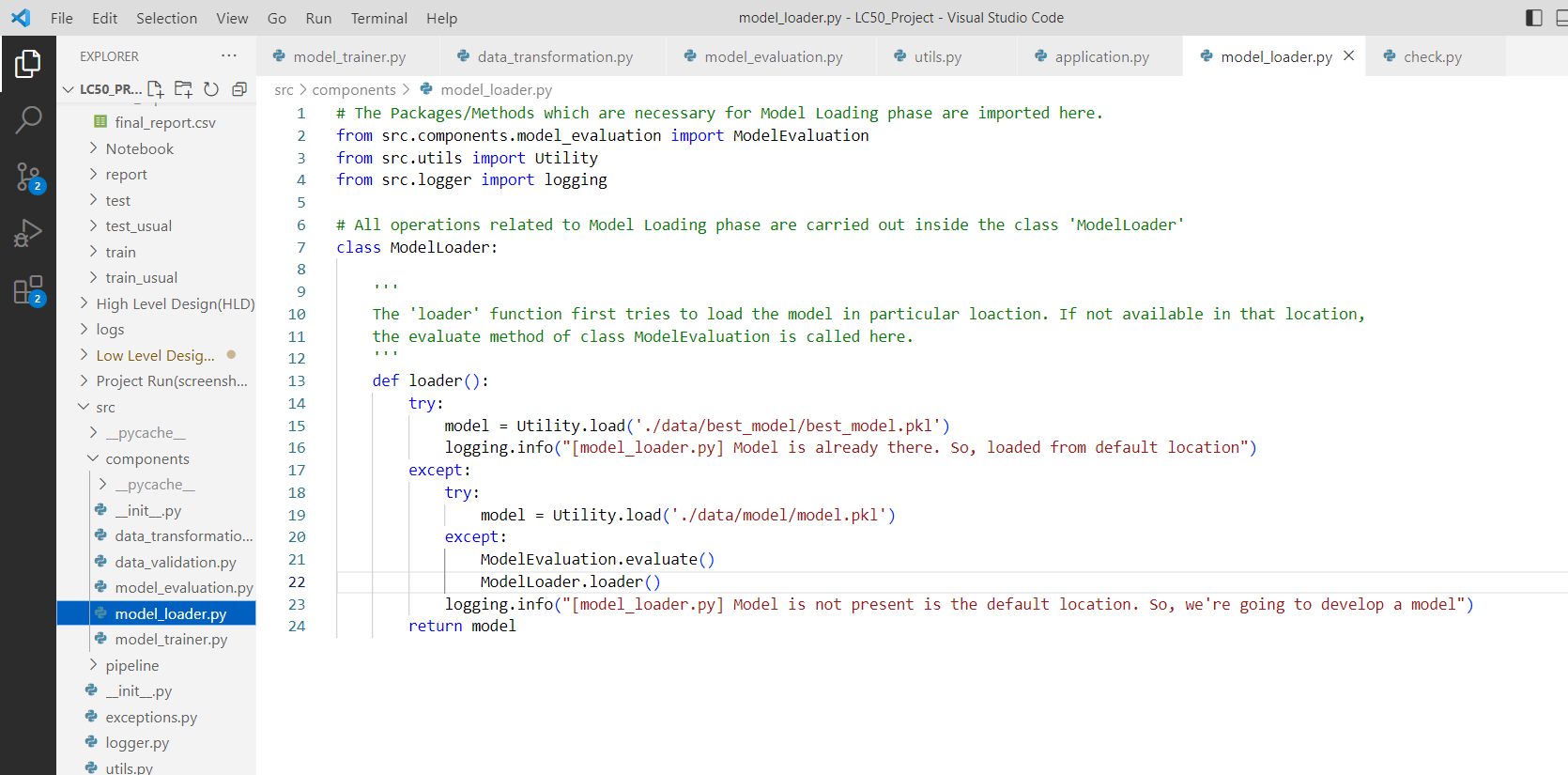
The function ‘custom\_model\_training’ is responsible for training the model if fails in traditional way of model training with the strong reason of huge difference between train and test score. The scenario may happen in some cases. So, we’ve to ready for those situation. The main objective of the function is to develop a generalized model. Both Un-tuned and Hyper-parameter tuned details will be recorded in the ‘final\_report.csv’. The ‘final\_report’ includes the informations of Model Name, Train Score and Test Score, ‘Depth(Hyper-paramter)’, ‘Iterations(Hyper-paramter)’ and ‘Learning\_Rate(Hyper-Parameter)’.

**3.6.9 Filtered Report**

****

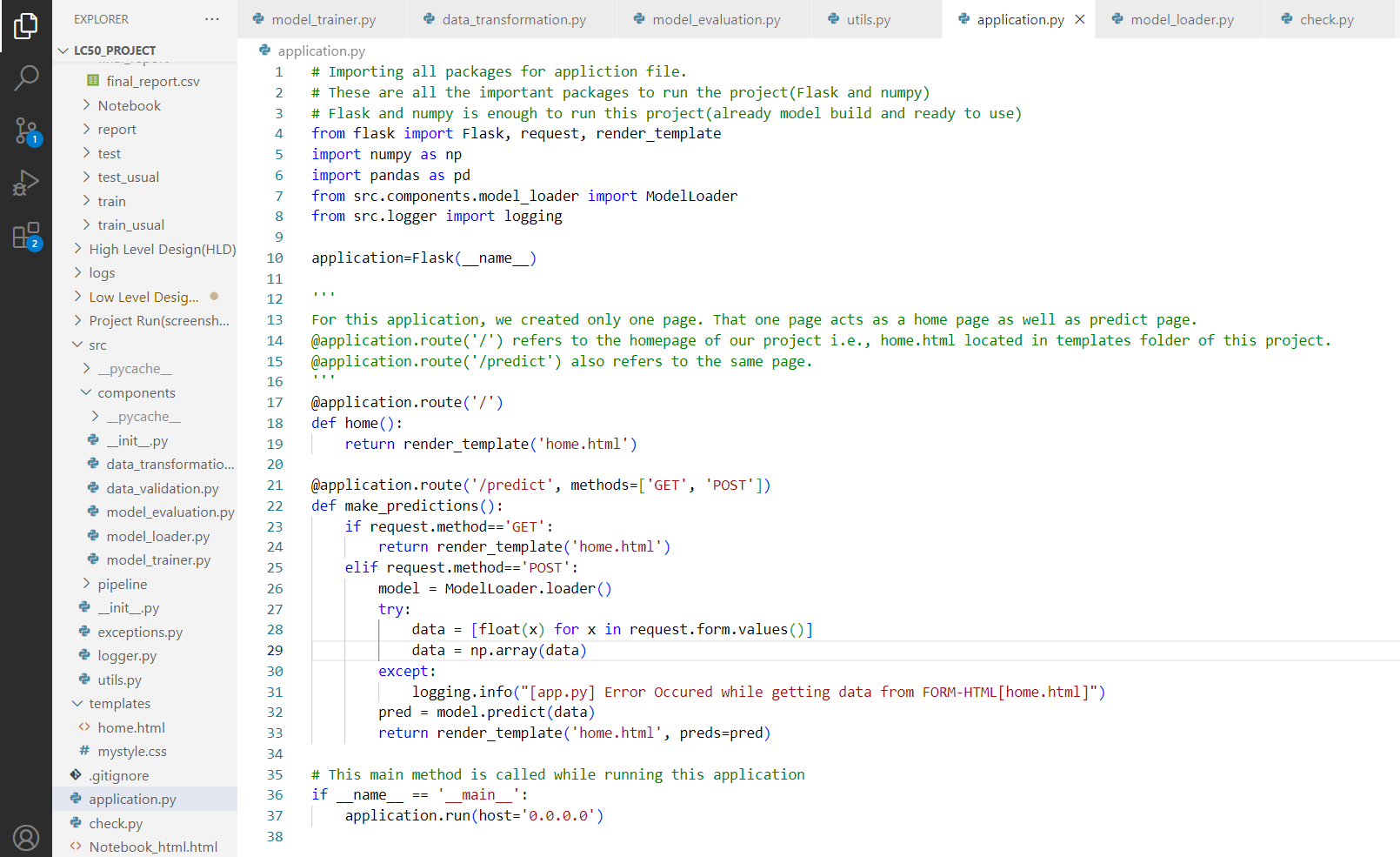
The ‘filtered\_report’ function is responsible for picking the hyper-parameters from the ‘final\_report.csv’ file which was generated in the function ‘custom\_model\_training’. The ‘final\_report’ file read from the particular dierctory. The Score Difference i.e., difference between train score and test score is calculated. The Score difference which is lower than zero were not considered because the training score must be equal or greater than the test score. If not, it is an underfitted model. After removing those negative values from the report, the report is arranged in an descending order with respect to test score. Now, the first record having a highest test score, meanwhile the train score is greater than test score. This results in obtaining the best parameters for optimal fit. In this project, we got an train score of 0.88 and test score of 0.86. This is much better than traditional output i.e., train score of 0.94 and test of 0.6. Finally, the filtered report is saved in a desired location and message were logged.

**3.7 Model Loader**

****

The necessary packages for ‘ModelLoader’ class will be imported. The ‘loader’ function first tries to get ‘best\_model’ from the desired location. If not, ‘loader’ checks for ‘model’. If there in no best\_model and model in the desired folder, the ‘evaluate’ function in ‘ModelEvaluation’ class is called to generate the desired model from the dataset. Then, loader is called again and any of the model is assigned to the variable ‘model’ and returned.

**3.8 Application**

****

The necessary packages like flask, model\_loader, etc., were imported. Then, inherting all attributes and methods in the class Flask, is assigned to an object ‘application’. The front-end design is developed and stored in a ‘templates’ folder. The home page of the project and prediction page using same template(home.html). After getting all inputs from the textbox and button click action, the prediction is shown is the new page which was rendered in the function ‘make\_prediction’. The ‘model\_loader’ is called and assigned to the variable ‘model’. The inputs from the front-end is received via ‘request’ method. Then, those data were converted into numpy array and passed to the predict method of ‘model’. Then it is returned to the ‘home.html’ page with the predicted result with the help of ‘render\_template’ method.

**4. Test Report**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case Description** | **Pre-Requisite** | **Expected Result** | **Actual Result** |
| To check the working condition of URL | URL of the project and web browser | It should work in every platform. | Working perfectly in all opeating system like Android, ios and mac. |
| To check the working condition on all web browser. | URL of the project and multiple web browsers | It should work in all web browser. | Tested in browser like chrome, safari and Edge. |
| To check the behaviour of the application with wrong input | URL of the project and web browser | It should return input error | Returns as Input Error |
| To check the output prediction | URL, web browser and desired inputs | It should return a output prediction in the web page | Returns prediction in the web page |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |