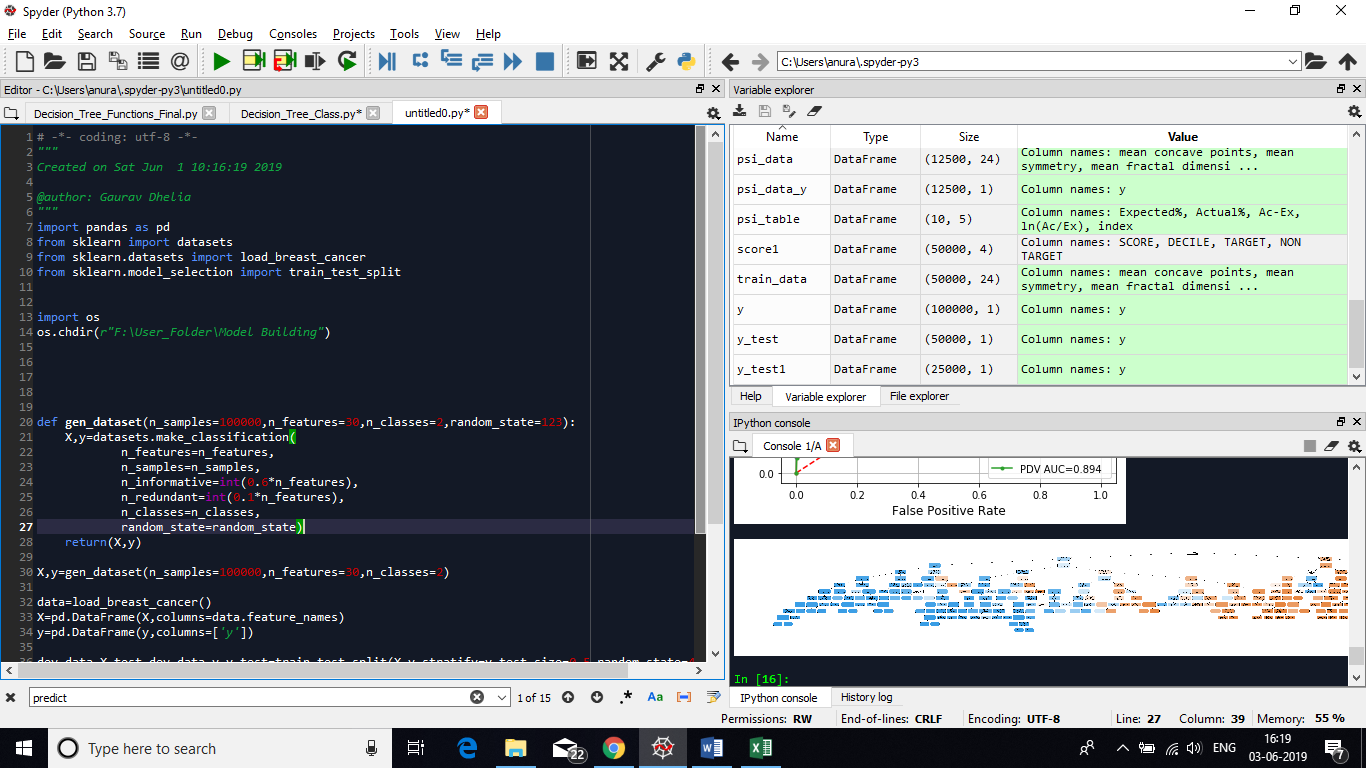
Functionalities of the SVM Regressor module:-

* It splits the development data into train and test, based on the user’s choice and the fraction specified by the user.
* It takes care of categorical columns i.e. create dummy variables for each category.
* It fills the null values of categorical variables by ‘MISSING’. It converts all the categorical variables to uppercase and strips the white spaces present in cases of string.
* It normalizes all the continuous (numerical) variables of the dataset. It also takes the natural logarithm of the dependent variable before making predictions and converts it back by exponent before giving any outputs. This is done to ensure the faster performance of the regression model.
* It builds the model with “C”=1.0, “gamma”=’scale’ and “kernel”=’rbf’ as the default value of the hyperparameters.
* It performs hyperparameter tuning to determine the best combination of parameters which would provide a best fit to the model. The user can opt if he/she wants to perform hyper parameter tuning for the specified variables. The hyperparameters to be tuned for this algorithm are “C”, “kernel” and “gamma”, the pool of values for which are [0.1,0.5,1,3,5], ['linear','rbf'] and [0.01,0.05,0.1,0.5,1.0] respectively by default.
* It fits SVM Regressor model on the training data and creates bands of 20 percentiles each. It then counts the median of the values in each bin for the dependent variable in the training dataset. It also scores the Out Of Sample, Out Of Time and Pre-deployment data and calculates R square, Adjusted R square, Mean Squared Error and Root Mean Squared Error for all these datasets.
* It calculates the univariate distribution (count,min,P0.5,P1,P2.5,P5,P10,P20,P25,P30,P40,P50,P60,P70,P75,P80,P90,P97.5,P99,max,missing%) for each dataset.
* It calculates the correlation co-efficient among all independent variables.
* It scores the PSI data and gives the median of values in each bin, after classifying them as per the percentile bins created by the dependent column of the training data.
* It saves all of the above mentioned results in an excel file along with the order of variables which would be used to score new data in future (ORDER SHOULD NOT BE CHANGED). These things are saved at the location provided by the user with the project name (also provided by the user), without any human intervention.
* It also saves the PSI graph created between the expected values obtained by the training data and the actual values obtained by the psi data (in the excel file and as a separate .png file) and the trained model in serialized form for future use in the same location without any human intervention.
* All the files in the provided location are saved such that the file names are followed by the time of the system at which the program is executed. This is done to avoid overwriting of the files when the program is executed multiple times.
* Attached below is a sample output which the code would generate.

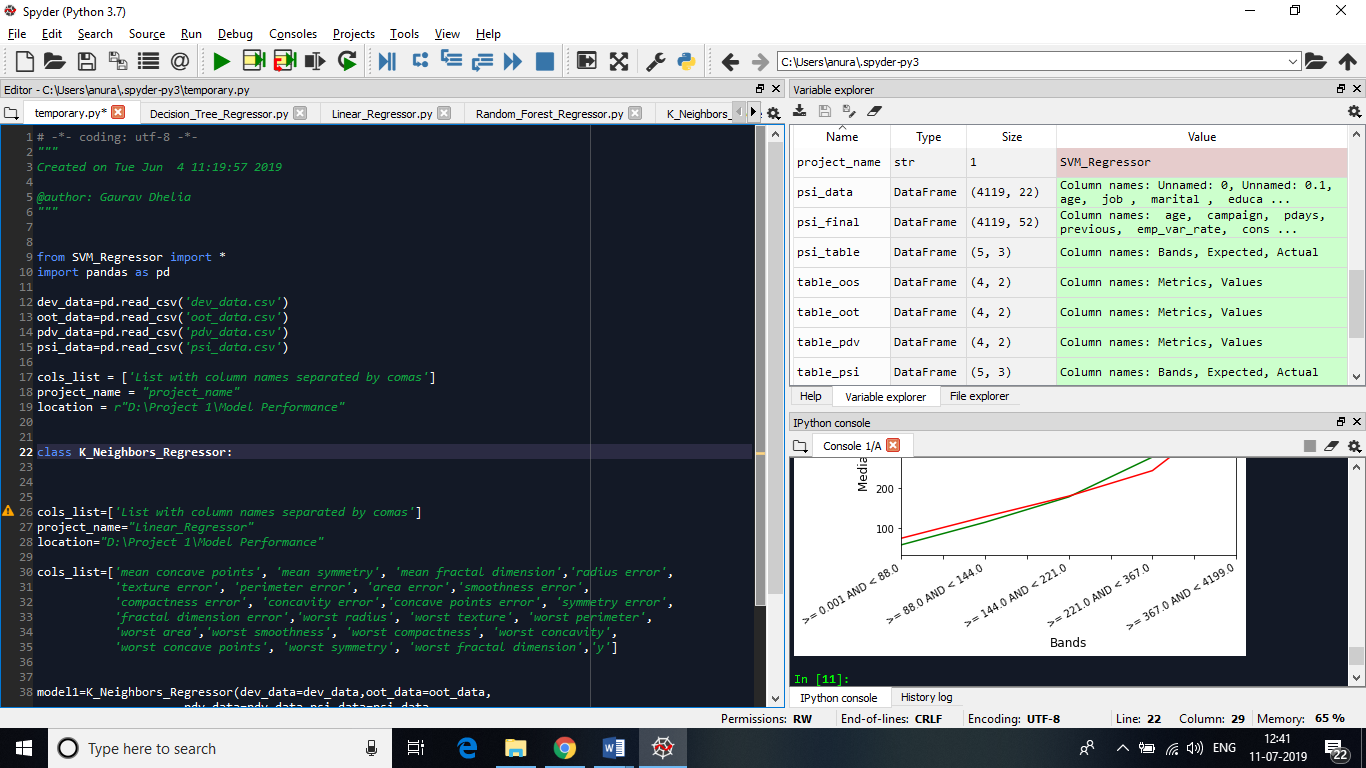
Steps for using the SVM Regressor module:-

* Open any Integrated Development Environment (IDE) to run python codes. Eg:- Spyder, Jupyter Notebook, etc.
* Set your working directory. Working directory is where all your data and codes reside.

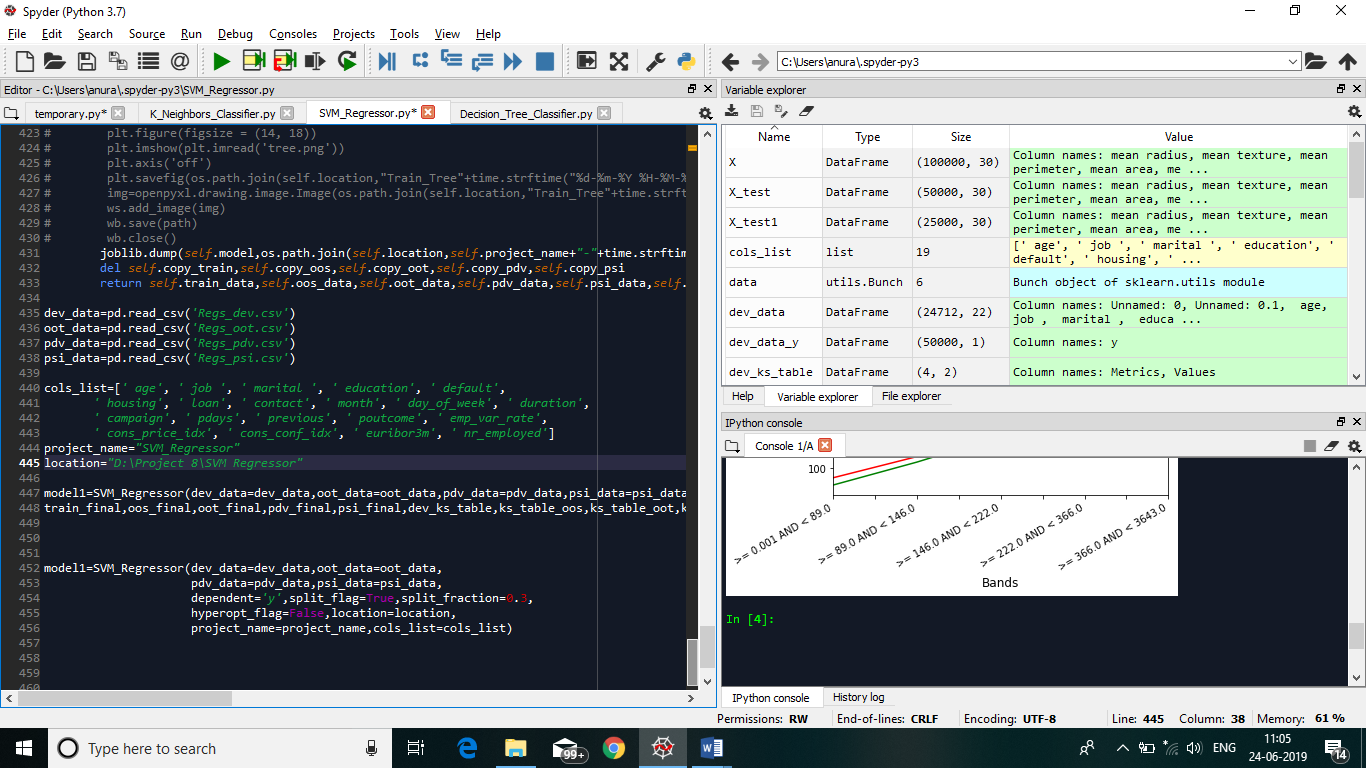


You need to specify your working directory in quotes as shown above.

* Replace F:\User\_Folder\Model Building with your working directory
* Copy SVM\_Regressor.py in your working directory folder.
* Import the SVM\_Regressor module. Enter the names of all the .csv files containing the data of different time frames- Out of Sample (if any), Out of Time, Pre deployment and Post Stability, and specify your project name, location where you want your results to be saved and the list of the column names for which the model has to be developed, as shown.



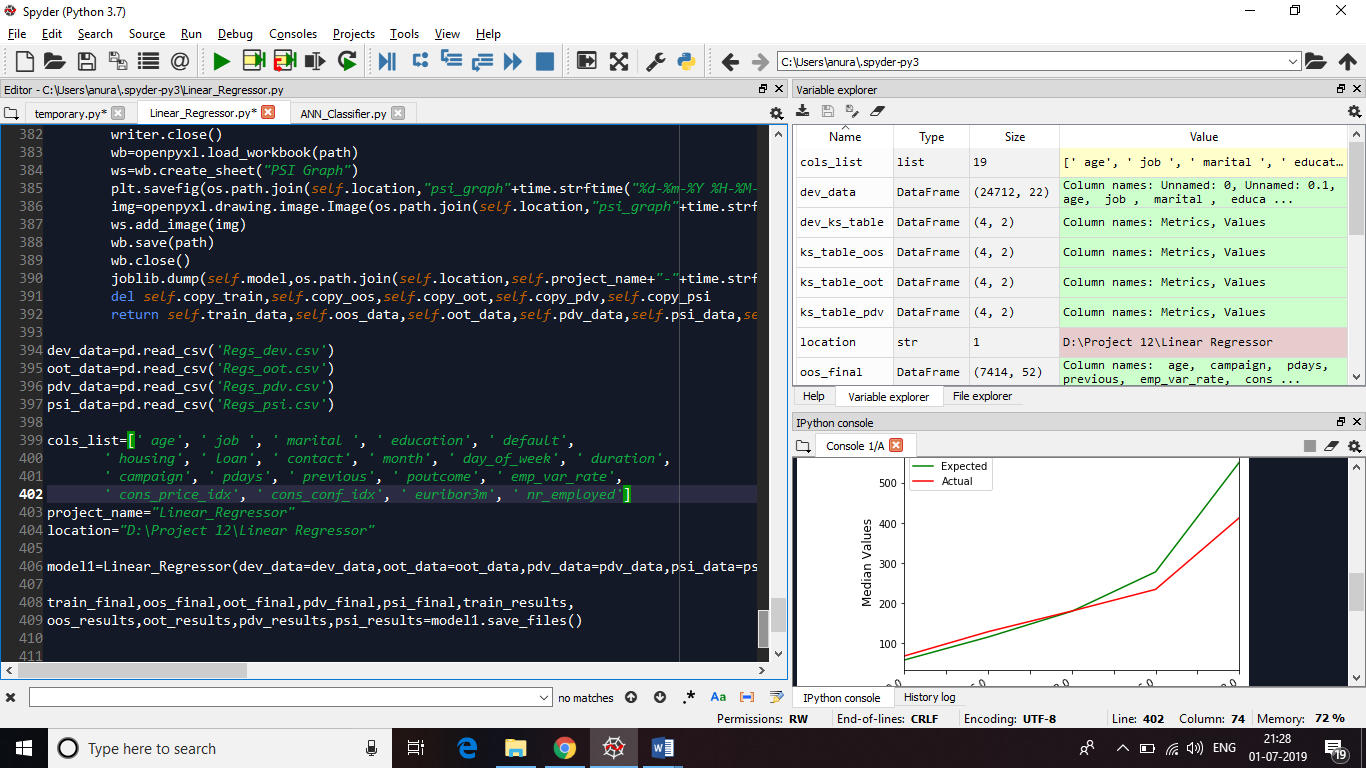
* Create an instance of class SVM\_Regressor and assign it to a variable (model1) as shown below):



It takes the following inputs:

1. dev\_data:- Specify the name of your Development dataset (as assigned to the dataframe of the .csv file)
2. oos\_data:- Specify the name of your Out Of Sample dataset (as assigned to the dataframe of the .csv file)
3. oot\_data:- Specify the name of your Out Of Time dataset (as assigned to the dataframe of the .csv file)
4. pdv\_data:- Specify the name of your Pre Deployment dataset (as assigned to the dataframe of the .csv file)
5. psi\_data:- Specify the name of your Population Stability Index dataset (as assigned to the dataframe of the .csv file)
6. dependent:- Specify the name of your target variable (dependent) in single/double quotes.
7. split\_flag:- Boolean value to specify whether you want to split your development data into development and OOS data. If True it splits the data into development and OOS using stratified sampling. If False, it uses the entire development data without splitting. It is set to False by default.
8. split\_fraction:- Float value to specify the fraction of split of the OOS data. It's value is passed only if the split\_flag is set to be True.
9. hyperopt\_flag:- Boolean value to specify whether you want to tune your hyperparameters. If True it uses Grid Search CV to find the best parameters. If False it uses the default parameters. It is set to False by default.
10. location:- The path of the folder where you want your results to be saved, entered in between single/double quotes.
11. project\_name:- Specify the name of your project as mentioned in previous step.
12. cols\_list:-Specify the name of your columns name list created while importing the SVM\_Regressor module. This column list also contains the name of the dependent variable.

* Using the class instance (model1 as defined above), call the save\_files() function as shown below:



By calling this function, the complete process of developing the model and saving of all the reports in the specified location is performed at once.

* The save\_files() function returns the following results:

1. train\_final:- The final train data after all the processing has been done to it.
2. oos\_final:- The final oos data after all the processing has been done to it.
3. oot\_final:- The final oot data after all the processing has been done to it.
4. pdv\_final:- The final pdv data after all the processing has been done to it.
5. psi\_final:- The final psi data after all the processing has been done to it.
6. train\_results:- The table created on the training data containing all regression metrics.
7. oos\_results:- The table created on the oos data containing all regression metrics.
8. oot\_results:- The table created on the oot data containing all regression metrics.
9. pdv\_results:- The table created on the pdv data containing all regression metrics.
10. psi\_results:- The table created between the expected values obtained by the training data and the actual values obtained by the psi data

* For more information to the class SVM\_Regressor, the user can look at the ‘Docstring’ after executing the program, which contains a description of all the functions contained within the class. It can be accessed by typing the following as shown, in the IPython Console of Spyder.

