

User Guide-Automated ML Pipeline

Mayank Kumar



Sections:

Section 1: Background & overview

Section 2: ML Pipeline Workflow

Section 3: ML Pipeline Modules

Section 4: Using ML pipeline in various scenarios

Section 5: Best Practices



Background:

Machine learning is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention. Machine learning is being used in a lot of areas like classification, computer vision, pattern recognition, natural language processing etc. Scope of this document is limited to the usage of machine learning for classification problems.

Classification refers to the prediction task where we have a set of variables and using it, we calculate probability for each possible outcome (class). This is all about finding a relationship between independent variables and target outcome. Traditional modeling techniques like logistic regression and decision tree help to explore linear relationships between independent variables and target outcome, but these techniques are not very effective when we want to capture non-linear relationships in the data.

Adoption of machine learning algorithms has increased in last 10 years. This is mainly due to availability of infra to train the models. Some of the most popular ML algorithms are GBM, xgboost, LightGBM and Catboost.

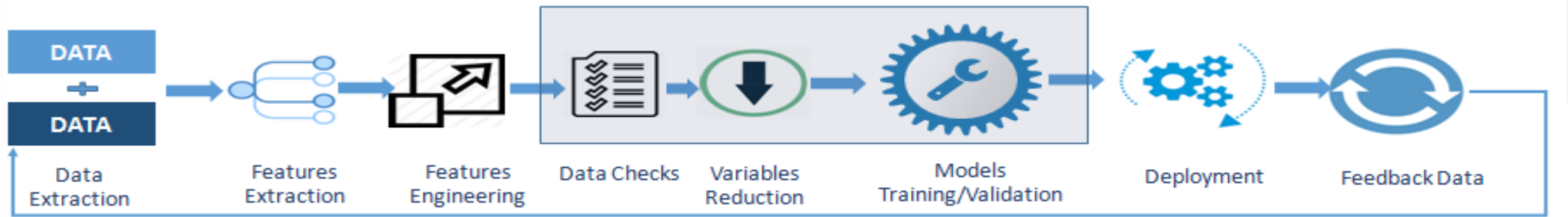
ML Pipeline –Overview:

I have developed automated ML pipeline. When I say automated ML pipeline, it's a set of modules which can help to do end-to-end modeling with minimal human intervention.

Three main objectives for building this automated ML pipeline:

1. Bring standardization in the process of building machine learning models
2. Recalibrate the models faster. If performance of the model goes below certain threshold, this automated ML pipeline can help to recalibrate the model faster.
3. Best-in-class ML algorithms available on a single click

Model Life-cycle:



When I talk about model life-cycle, it has eight broad stages. Scope of ML pipeline is limited to three blocks highlighted in blue. ML pipeline handles data checks, variables reduction and model training/validation. When this is integrated with automated codes for data extraction/features creation, it helps to reduce TAT significantly.

ML Pipeline – Workflow:

ML Pipeline has four modules. Model developer will have flexibility to use these modules in isolation or using all the modules in one go.

Modules:

1. Univariate and Bivariate
2. Variable selection
3. Grid search (Xgboost, lightgbm, catboost- Building 1000+ models)
4. Comparion of models and select the best model

Going one level down, ML pipeline performs the following tasks at the backend.

1.	Create Datasets: Development, OOS, OOT, PDV Exclude categorical variables having categories greater than max_elem parameter (default value is 25). Excluded variables will be shown in variable reduction report under stage “High_Categories_Var_Filter”
2.	Run bivariate analysis. Select the variables having IV > 0.02
3.	If correlation between two variables > 0.9, choose the variable having higher IV
3.1.	If Cramer’s V between two categorical variables > 0.5, choose the variable having higher IV
4.	In case of categorical variables, apply one-hot encoding
4.1	Pass sample weight. This is to handle imbalanced dataset
5.	Every model (xgboost,lightgbm and catboost) mentioned is executed and checked for its feature importance and a rank is created(based on feature imp). Correlation is checked between top 10 variables and the variable having correlation value of >0.4 (correlation cutoff) and having a higher rank (low feature importance) is dropped. One variable at a time is getting dropped, so that change in feature imp after dropping correlated var can also be taken into consideration. This loop is getting executed max 6 times. Idea here is to drop low feature importance variables from top 10 high correlated variables. 5.1 One hot encoded categorical variable if not present in OOS/OOT/PDV will be removed and reported in variable reduction report under stage “Inconsistent_Categories_Across_OOS_OOT_PDV”. 5.2 Check for variables having correlation >0.4 (Correlation cutoff) among all remaining variables. Then we pick the variables having highest correlation (for eg. If var 1 is having high correlation with var2 (0.8), var3(0.7), var4(0.6) then we pick the pair Var1 and Var2. Then we compare the ranks of original variables & correlated variables and compute the gap between ranks of those 2 pairs of variables (var1 with feature imp rank 1 and var2 with feature imp rank 20 then gap will be 19). Higher the gap, lesser the impact of correlated variable in the model. Then we drop max of 10% of correlated variables above cut-off or 10 variables in one go. If the total number of variables > 30 (max_feat parameter) after this step, then run the model, calculate feature importance and select top 30 (max_feat parameter) variables to move ahead with the next steps.
6.	Train the models on all time frames and calculate feature importance and rank of the variable on all time frames. If any variable has zero feature in any period, drop that variable.
6.1	Train the models on all time frames and calculate feature importance and rank of the variable on all time frames. If any variable has a rank difference> 10 as compared to dev dataset, then drop that variable too.
7.	Each remaining variable gets a score-based PSI (population stability index) Variable with high score means less stable. A model is also trained on these variables and the top variables based on highest feature importance and least score (most stable) are selected as final variables before passing it for grid search. Variables having PSI score greater than psi_cutoff (default value 20) are dropped in this stage. Note: a. It may be possible that OOS/OOT/PDV can have zero population distribution for some of the bins for some of the variables. To make sure PSI LOG function doesn’t throw error, we consider 0.000001 as a compensating factor. b. Separate MISSING category bin is created for missing records.
8.	Grid search is applied on the final variables
9.	For each algorithm, out of all the combinations in the grid search, the best model is chosen based on if the KS difference between time frames is <15% and amongst them who has the highest KS in dev time period. Once we get one best model per algorithm, the best final best model is chosen based on who has highest Dev KS.
9.1	In cases where any algorithm doesn’t produce a combination in the grid search who has a KS difference of less than 15% then for that algorithm the combination having the minimum difference in KS is chosen as the best model. Once we get one best model per algorithm, the best final best model is chosen based on who has highest Dev KS.
10.	Once the best model is selected along with-it best parameter and final list of variables, A last iterations gets executed to check if any of the variables have zero feature importance. If there is then that variable is dropped and the model is trained again. If not then that is the final model.

ML Pipeline – Modules:

Module 1: Univariate and Bivariate Module

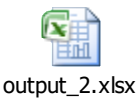
Description : This module will help the user get the univariate and bivariate analysis.
Usage : To run this module, execute the below line and save the results in a variable.

```
Variable = univariate_bivariate(data,target, vars_exc,loc=None)
```

Parameters:
data – your developement data set (eg X_train)
target - your target variable name (eg 'Default')
vars_exc - A list of variables that you do not need (eg ['mis_Date', 'id', etc])
loc – location/path where you want the output to be stored.
Note : User has to pass the path in the mentioned format below or else the pipeline will throw an error
eg ('I:/Pool User Folder/Mayank/ML PIPELINE/filename.xlsx'
Note:Excel File name at the end of the path has to be mentioned

Output :
Univariate_numeric :univariate analysis for numeric variables.
Univariate_character :univarriate analysis for character variables.
Cardinality_1_and_high_NA: variables which have cardinality 1 and missing values more than 90%.
Bivariates : Bivariate analysis
overall_IV: variable names along with its IV value
overall_IV_description: A distribution of the IV values which will help in selecting the best IV cutoff.

Sample Output:



Module 2: Variable reduction Module

Description: This module performs variables reduction using the following set of guidelines.

Below are the steps that have been incorporated in this module.

- Run bivariate analysis. Select the variables having IV > 0.02
- If correlation between two variables > 0.9, choose the variable having higher IV
- If cramer's V between two categorical variables > 0.5, choose the variable having higher IV
- In case of categorical variables, apply one-hot encoding
- Pass sample weight. This is to handle imbalanced dataset.
- Every model (xgboost, lightgbm and catboost) mentioned is executed and checked for its feature importance. These models run having max Depth=3, n_estimators =100 and random_state=42. After the feature importance is calculated, only the features above the importance threshold are selected. By default, the importance threshold is 0.01, but the user can change this.
- Check top 10 variables. Remove variables which are correlated (correlation > 0.4) with these top 10 variables and having very low feature importance in terms of rank (diff rank > 10). This is an iterative process which happens while executing all 3 algorithms. Execution continues till the variable count reaches ~25-30.
- After the above stage, these 25-30 variables are checked on all three datasets (Development, OOS, OOT) along with its feature importance. Based on population and bad rate shift a score is calculated. The final variables are selected based on the lowest scores which include most stable variables.

Reports:

- Variables dropped basis high missing %
- Variables dropped basis low IV
- Variables dropped basis high correlation (Numeric variables)
- Variables dropped basis high Cramer's V (Categorical variables)
- Top variables for each algorithm (xgboost/LightGBM/Catboost)



Grid search module:

Description: Grid search is the process of scanning the data to configure optimal parameters for a given model. It iterates through every parameter combination and stores a model for each combination. It is important to note that grid-searching can be extremely computationally expensive and may take your machine quite a long time to run.

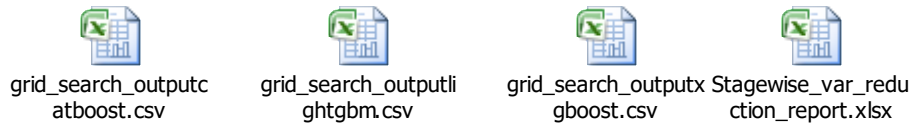
Grid search is done for all the 3 mentioned algorithms so as to get the best hyper parameters. Users can also make use of custom grids of different parameters for one or all of the 3 algorithms. This function also calculates KS for all time frames and for every combination of the parameters.

Note: If no custom grid search is mentioned then, the pipeline will consider the following parameters:

For xgboost: {'max_depth':[1,2,3,4], 'eta':[0.01,0.03,0.05,0.07,0.1,0.15,0.2],
'n_iteration':[25,50,75,100], 'colsample_bytree':[0.5,1]}

For lightgbm: {'max_depth':[1,2,3,4], 'learning_rate':[0.01,0.03,0.05,0.07,0.1,0.15,0.2],
'num_iterations':[25,50,75,100], 'bagging_fraction':[0.5,1], 'max_bin':[127,255]}

For catboost : {'depth':[1,2,3,4], 'learning_rate':[0.01,0.03,0.05,0.07,0.1,0.15,0.2],
'iterations':[25,50,75,100], 'border_count':[127,254]}

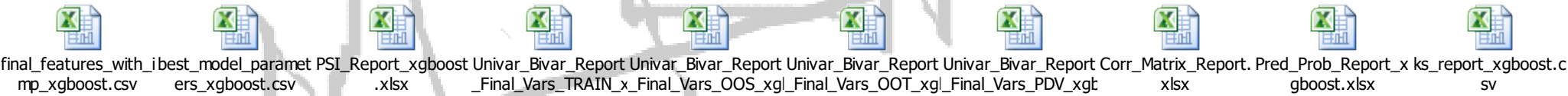


Model comparison module:

Description: Among the three algorithms, the best one is selected based on highest KS and consistency along all time frames.

Reports:

- The best model along with its final features and variables importance across TRAIN, OOS, OOT & PDV
- The best model along with its optimal parameters
- PSI report for all variables in the final model across TRAIN, OOS, OOT, PDV, both at variable bin level and variable level.
- Univariate-Bivariate report of all variables in the final model across TRAIN, OOS, OOT, PDV.
- Correlation matrix of all variables in the final model across TRAIN, OOS, OOT, PDV.
- Predicted probability report for all variables in the final model across TRAIN, OOS, OOT, PDV.
- Final model KS report across TRAIN, OOS, OOT, PDV.



Using ML pipeline in various scenarios:

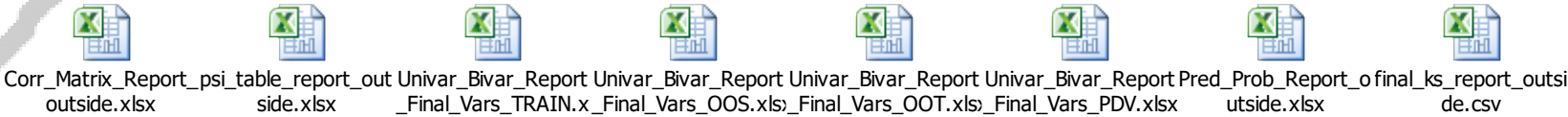
This section covers usage of ML pipeline in four most prominent scenarios.

NOTE:

1. Make sure you are using less than 5 categories in categorical variables. If you need to use all categories, please encode that variable using some encoding technique and then pass the data in the pipeline.
2. This code does not help in creating segments and it is advisable to use sampled data in place of the entire data set for faster execution.
3. There is no need to pass “scale_pos_weight” in the pipeline as a parameter as the pipeline uses sample weights which takes care in case of fraud scorecards.
4. The model file that gets exported is a pickle file.
5. If the user wants to runs the model manually using the optimal parameters and final variables, the user can do so by just loading the model file.
6. If you have categorical variables in the development data, make sure all those categories are present in all time frames.
7. Sample script to run the auto-ml pipeline is attached below.
8. In case of CATBOOST model being selected as a best model and user wants to use the saved model file to predict the probabilities, run KS report etc. outside auto-ml pipeline then user should pass the final variables in the same sequence as per the auto-ml pipeline trained catboost model. The auto-ml pipeline final variable sequence is stored in the “final_feat_seq_catboost.csv” file. If user changes the order of the variables, then he/she may get different results.



9. If user already has a trained model file and wants to export below reports then he/she can use generate_reports_outside.py script in order to do so. Use has to modify user input parameters in the user inputs section. Use will get below reports as output.
 - a. PSI report for all variables in the final model across TRAIN, OOS, OOT, PDV, both at variable bin level and variable level.
 - b. Univariate-Bivariate report of all variables in the final model across TRAIN, OOS, OOT, PDV.
 - c. Correlation matrix of all variables in the final model across TRAIN, OOS, OOT, PDV.
 - d. Predicted probability report for all variables in the final model across TRAIN, OOS, OOT, PDV.
 - e. Final model KS report across TRAIN, OOS, OOT, PDV.



10. Considering deployment related limitations in real time environment, I am keeping hyper parameter of maximum number of trees to 100 only. If user is planning to run the auto-ml pipeline in batch mode and not in real time, user may want to tweak the parameter.

Exhaustive list of model development class user parameters & their definition:

Depending on use case, you may want to tweak parameters like Final_corr_max, psi_num_bins, IV_min, max_models, grid_search_param_xgb, grid_search_param_gbm, grid_search_param_cat.

model_development(uniq_key, dev, target, vars_exc, oos, oot, bad, psi_num_bins=5, pdv=None, loc=None, IV_min=0.02, cor_max=0.9, cramer_max=0.5, method=["xgboost"], NA_cutoff=0.8, IV_Data=None, max_feat=30, Final_corr_max=0.5, max_elem=25, final_model_elem=10, grid_search_param_xgb=None, grid_search_param_gbm=None, grid_search_param_cat=None, max_models=1500, model_selection_seed=1255, model_random_seed=42, run_till_bivariate=False, psi_cutoff=20)

- uniq_key :- Unique key in the input dataset (string)
- dev :- Development data set (pandas dataframe)
- target :- target variable name (list)
- vars_exc :- variables that user would like to exclude (list)
- oos :- Out of sample data set (pandas dataframe)
- oot :- Out of time data set (pandas dataframe)
- bad :- Should be equal to 1 as it stands for bads (int)
- psi_num_bins :- Number of variable bins for PSI report computation (int)
- pdv :- pre-deployment validation data set (pandas dataframe)
- loc :- Location/path where user want the reports to be stored. Filename should be provided as a part of path. (string)
- IV_min :- IV value cutoff to be considered as a part of variable reduction process (float)
- cor_max :- Pearson correlation cutoff to be considered as a part of variable reduction process (float),
- cramer_max=Cramer's V cutoff to be considered as a part of variable reduction process,
- method :- list of algorithms to be considered out of xgboost, catboost, lightgbm (list)
- NA_cutoff :- missing value % cutoff (float)
- IV_Data :- Biivariate report (pandas dataframe)
- max_feat :- maximum number of variables to be considered in step 5.2 of the workflow (int)
- Final_corr_max :- Pearson correlation cutoff in step 5.2 of the workflow (float)
- max_elem :- Maximum number of variable level to be considered for categorical variables. Variables having categories greater than max_elem will be excluded (int).
- final_model_elem :- Upper cutoff on number of final shortlisted variables
- grid_search_param_xgb :- Hyper parameters for grid search of XGBoost (dictionary with keys as hyper-parameter name and value as value of the hyper-parameter)
- grid_search_param_gbm :- Hyper parameters for grid search of LightGBM (dictionary with keys as hyper-parameter name and value as value of the hyper-parameter)
- grid_search_param_cat :- Hyper parameters for grid search of Catboost (dictionary with keys as hyper-parameter name and value as value of the hyper-parameter)
- max_models :- Maximum number of candidate models to be created for each of the algorithm (int)
- model_selection_seed :- Seed for selecting sample of candidate hyper-parameters (int)
- model_random_seed :- Seed used while training the model (int)
- run_till_bivariate :- Flag whether to run entire auto-ml pipeline or only till univariate-bivariate(Bool)
- psi_cutoff :- PSI threshold above which the variables will be removed in stage 7 of the workflow (int)

Note:

1. model development class arguments specify default values.
2. User parameters definition specifies expected data type in parenthesis.

Scenario 1

Description: User has four datasets- dev, oos, oot and pdv. User wants to try all configured algorithms for developing ML model.

Usage: The user will just have to run the below line of code.

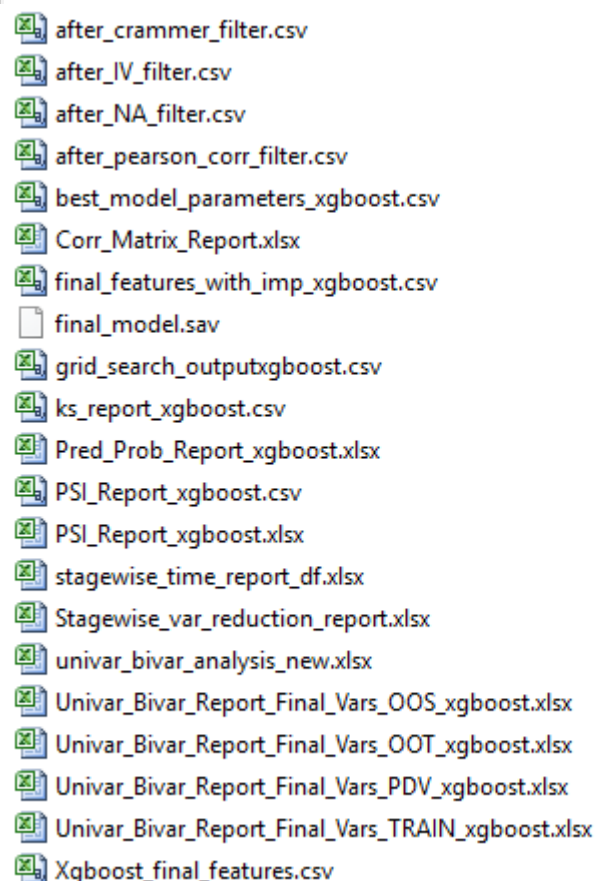
```
model_development(uniq_key=uniq_key,dev,target, vars_exc,oos,oot,bad=1,pdv=None,loc=None,IV_min=0.02,
method=["catboost","xgboost","lightgbm"],NA_cutoff=0.8)
```

Parameters:

- Uniq_key – Unique key in the input dataset
- dev - Developement data set (eg X_train)
- target - Target variable name (eg ['Default'])
- vars_exc - A list of variables that user would like to exclude(eg ['mis_Date', 'id', etc])
- oos - Out of sample data set (eg X_test)
- oot - Out of time data set (eg oot)
- bad – should be equal to 1 as it stands for bads
- pdv – your pre-deployment data set
- loc - location/path where you want the reports to be stored.
eg ('I:/Pool User Folder/Mayank/ML PIPELINE/filename.xlsx')
- Note : filename will consist of univariate and bivariate reports and others modules will have thier own excel sheets. It is compulsory for the user to specify the filename here.
- IV_min – IV value cutoff, by default it is 0.02
- method – a list which contains the methods that can be used.Here the user can mention 1 method or all 3.
Note: If the user does not specify anything then, Xgboost will be taken as default.
- NA_cutoff – Na value % cutoff. By default it is 80%.

Output :

1. Univariate and Bivariate reports.
2. Reports which consists of the variables selected after each variable reduction filter.
3. Reports for grid search(Xgboost, lightgbm, catboost)
4. Reports on the final features selected for each model
5. Reports on the best model among the 3 along with its optimal parameters.
6. Best model is also exported.
7. PSI report for all variables in the final model across TRAIN, OOS, OOT, PDV, both at variable bin level and variable level.
8. Univariate-Bivariate report of all variables in the final model across TRAIN, OOS, OOT, PDV.
9. Correlation matrix of all variables in the final model across TRAIN, OOS, OOT, PDV.
10. Predicted probability report for all variables in the final model across TRAIN, OOS, OOT, PDV.
11. Final model KS report across TRAIN, OOS, OOT, PDV.



- after_crammer_filter.csv
- after_IV_filter.csv
- after_NA_filter.csv
- after_pearson_corr_filter.csv
- best_model_parameters_xgboost.csv
- Corr_Matrix_Report.xlsx
- final_features_with_imp_xgboost.csv
- final_model.sav
- grid_search_outputxgboost.csv
- ks_report_xgboost.csv
- Pred_Prob_Report_xgboost.xlsx
- PSI_Report_xgboost.csv
- PSI_Report_xgboost.xlsx
- stagewise_time_report_df.xlsx
- Stagewise_var_reduction_report.xlsx
- univar_bivar_analysis_new.xlsx
- Univar_Bivar_Report_Final_Vars_OOS_xgboost.xlsx
- Univar_Bivar_Report_Final_Vars_OOT_xgboost.xlsx
- Univar_Bivar_Report_Final_Vars_PDV_xgboost.xlsx
- Univar_Bivar_Report_Final_Vars_TRAIN_xgboost.xlsx
- Xgboost_final_features.csv

Above are the files that will get generated in the location that the user specifies.In the above **example** output_2 is the excel file name that the user has to mention while

giving the location.

Best_model_parameters[method_name] file will tell the user which is the best model along with its optimal parameters.

final_features_with_imp[method_name] will give the user the final features the model has selected along with its variable importance

for example, best_model_parameterslightgbm (the best model selected is lightgbm and its optimal parameters will be stored in this file)

final_model.sav is best model that is exported and can be used later.

Note : dev,oos and oot is mandatory datasets that have to be passed in the function.PDV is optional.

By running this the user will get ~ 900 models for each method. It is advised to run this overnight as it will take time to compute everything.This has been tested for multiple data sets and if the user gets an error it is mostly because there will be something wrong in some variable.

Scenario 2

Description:

If the user wants to use the entire pipeline but already has a set of final selected features then the user can do so by following the below steps.

Usage:

First create a list a all the final selected variables and also include the bad flag in the list.

For example

```
final_Coulms = ['bad_flag','AQB', 'DPD_X_I3', 'DPD_X_I9',etc]
```

Then use only those columns in the data sets that are being passed as shown below.

```
model_development(uniq_key=uniq_key, dev=X_train[final_Coulms],target=tar, vars_exc  
=vars_exc,oos=X_test[final_Coulms],oot=oot[final_Coulms],bad=1,method=["lightgbm","catboost","xgboost"])
```

The outputs will be the same as compared to scenario one but will be made using only the final selected variables.

Scenario 3

Description:

If the user wants to run the entire pipeline but wants to make use of custom parameters for grid search for one or all three methods the user can do so by following the below instuctions

Usage:

```
model_development(uniq_key=uniq_key,,dev=X_train,target=tar, vars_exc = vars_exc,oos=X_test,oot=oot,pdv=pdv,loc=loc,bad=1,  
method=["catboost", "lightgbm", "xgboost"],
```

```
grid_search_param_cat= {'depth':[2,3,4,5], 'learning_rate':[0.01,0.03,0.05,0.07,0.1,0.15,0.2],  
    'n_estimators':[25,50,75,100], 'reg_lambda':[1,2,3,4]},
```

```
grid_search_param_gbm= {'max_depth':[2,3,4,5], 'learning_rate':[0.01,0.03,0.05,0.07,0.1,0.15,0.2],  
    'n_estimators':[25,50,75,100], 'bagging_fraction':[0.5,0.75,1], 'reg_lambda':[1,2,3,4]},
```

```
grid_search_param_xgb= {'max_depth':[2,3,4,5], 'learning_rate':[0.01,0.03,0.05,0.07,0.1,0.15,0.2],  
    'n_estimators':[25,50,75,100], 'colsample_bytree':[0.5,0.75,1], 'reg_lambda':[1,2,3,4]}}
```

The user can add or remove parameters in the above shown way.

Note : If more parameters are added then the combinations of models will increase which will take the pipeline more time to execute.

Output of this will be the same as compared to scenario 1

Scenario 4

Description:

If user wants to keep maximum number of final features to say 25 instead of default of 10, then user can do so by following below instructions:

Usage:

```
model_development (uniq_key=uniq_key,dev=X_train,target=tar,vars_exc=vars_exc,  
oos=X_test,oot=oot,pdv=pdv,bad=1,method=method,loc=loc,IV_min=0.02,NA_cutoff=0.8,  
    max_feat=50, This parameter decides how many features to keep at “High_correlation_top_to_remaining_vars” stage of variable reduction stages  
    final_model_elem=25 ## This is the parameter setting for maximum no of final features  
)
```

Note: max_feat should be set to higher than final_model_elem.

final_model_elem parameter is for setting MAXIMUM number of features user wants as the final features to be used for grid search and final output.

It will not guarantee the user that final features will be exactly 50. Final features can still be lower number say 10 or 5. This is because the variable resuction stages such as Zero_feature_imp_DEV_OOT_PDV and Rank_Diff_GT_10_DEV_OOT_PDV can remove many features in one go which satisfy the specific criteria in those stages and we don't have exact control over that in terms of number of features to remove in these 2 stages.

Scenario 5

Description:

If user wants to run auto-ml pipeline only for univariate-bivariate computation, then set parameter run_till_bivariate equal to True as shown below..

Usage:

```
final_out_test = model_development (uniq_key=uniq_key,dev=X_train,target=tar,vars_exc=vars_exc,
                                     oos=X_test,oot=oot,pdv=pdv,
                                     bad=1, psi_num_bins=10, method=method,
                                     loc=loc,IV_min=0.02,NA_cutoff=0.8,
                                     max_feat=50, final_model_elem=25,
                                     run_till_bivariate=True)
```

Scenario 6

Description:

If user wants the PSI report across say 5 bins instead of default of 10 of each of the final shortlisted variables, then set parameter psi_num_bins equal to 5 as shown below.

Usage:

```
final_out_test = model_development (uniq_key=uniq_key,dev=X_train,target=tar,vars_exc=vars_exc,
                                     oos=X_test,oot=oot,pdv=pdv,
                                     bad=1, psi_num_bins=5, method=method,
                                     loc=loc,IV_min=0.02,NA_cutoff=0.8,
                                     max_feat=50, final_model_elem=25,
                                     run_till_bivariate=True)
```

Errors that are usually encountered:

- 1) Access Denied error: To avoid this make sure you have write access to the folder path that you will be giving.
- 2) Variable importance does not match if you run the model outside the pipeline with same features and same parameters. To avoid this please make use of the same order of variables as provided in the final_features files. You will also need to pass sample_weight parameter in the fit function.

Best Practices:

Based on my experience of building the ML models over the period, below are the broad level steps user should follow during model development process. These steps may vary depending on use case, so you may also want to discuss with your respective TL.

Sr. No.	Stage	Step
0	Starting Point	ML Pipeline needs to be triggered after segmentation stage. Once you have segment level train, test, OOT, PDV datasets available, ML Pipeline can help to build the model faster.
1	Data Prep Stage	Ensure that you are passing meaningful features in ML pipeline. You should avoid passing categorical features with more than 10 levels (For Example-Postal code, City etc.). If you want to pass such variables, do some logical binning and then pass into the model.
		Before you run ML pipeline, ensure that you don't have any variable basis which bad flag has been created. (For example, If bad flag is created basis application status- A/R, then please don't use application status as a variable.
		Check with the stakeholder if they have any preference for any rolling time-window. ML pipeline will purely go by statistical indicators. For example, if count_cash_deposits_3M is powerful than cash_deposits_6M, then it will pick up count variable for three months. If your stakeholders want variables for six or higher time-window, then we should drop variables pertaining to one month/three months before running ML pipeline.
2	Getting Ready for ML pipeline	Be sure about correlation cut-off in stage 5.2 of variable reduction. Do you want to go with 0.4 or higher cut-off? Would recommend 0.5.
		Select maximum variables in final model? 10 or higher? (Parameter in auto-ml pipeline - final_model_elem)
		Check if Gridsearch hyper parameters to be changed. Want to restrict to 100 trees? In case of batch models, you may want to increase thresholds.
		Define uniq_key, vars_exc and target variables properly
3	Evaluate your final model	Are final shortlisted variables form ML pipeline making business sense.
		Check variable feature importance.
		Check for PSI scores for stability of features across timeframes
		Check for KS tables
		Decile level check
4	Manual Iterations	Revise the list of variables (Basis feedback from functional team)
		Don't run ML pipieline with all variables again. Tune model with final 30-40 variables.
		Finalize the model
		Play with the same technique which was giving the best model earlier (No need to run gridsearch again for all three methods)
		Try random grid search or bayesian grid search if you want. Document is attached in appendix section.
5	Score Bands-Finalization	Check if deciles can be clubbed to ensure that decile level no flips (Either combine deciles or create bands)
6	What to do if model is not stable?	Check bivariate report of final variables
		Consider using unstable variable (At the max two) by passing continuous vars in binned form
		Use DT or LR as an alternative

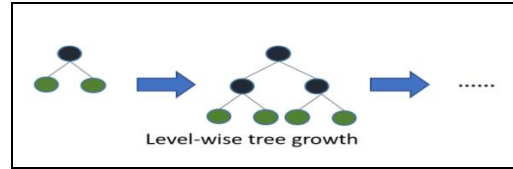
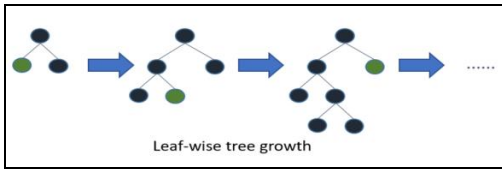
Appendix:

LightGBM

LightGBM stands for lightweight gradient boosting machines. It is prefixed as 'Light' because of its high speed. It was developed by Guolin Ke (Microsoft), and was presented in the proceedings of the 31st International Conference on Neural Information Processing Systems, 2017.

Like XGBoost, LightGBM is also a subtype/specific instance of the Gradient Boosting Decision Tree (GBDT) algorithm. The difference between XGBoost and LightGBM is in the specifics of the optimizations.

- LightGBM grows tree vertically while other algorithm grows trees horizontally, meaning that it grows tree **leaf-wise** while other algorithm grows level-wise. It will choose the leaf with max delta loss to grow. When growing the same leaf, leaf-wise algorithm can reduce more loss than a level-wise algorithm. Leaf wise split leads to increase in complexity, and may lead to overfitting. It can be overcome by specifying another parameter max-depth which specifies the depth to which splitting will occur.



- It uses histogram based algorithm, i.e., it buckets continuous feature values into discrete bins which leads to faster training and higher accuracy. Replacing continuous values with discrete bins also results in lower memory usage
- GOSS (Gradient based One Sided Sampling) – The essential motivation behind this method is that not all data points contribute equally to training; data points with small gradients tend to be better trained (closer to a local minima). This means that it is more efficient to concentrate on data points with larger gradients. GOSS retains instances with large gradients while performing random sampling on instances with small gradients.
- EFB (Exclusive Feature Bundling) - The essential motivation behind this method is the sparsity of features which means that some features are never non-zero simultaneously (mutually exclusive). LightGBM safely identifies such features, and bundles them into a single feature to reduce the complexity.
- Automatic handling of missing values - It ignores them during split finding, and then allocates them to whichever side reduces the loss the most. Though LightGBM does not enable ignoring zero values by default, it has an option called *zero_as_missing* which, if set to *True*, will regard all zero values as missing.
- It can handle the large size of data with a significant reduction in training time as compared to XGBoost. The significant speed advantage of LightGBM translates into the ability to carry out more iterations, and/or quicker hyper-parameter search.
- It also supports GPU learning.

XGBoost

Its name stands for eXtreme Gradient Boosting. It was developed as a research project at the University of Washington by Tianqi Chen and Carlos Guestrin. It belongs to a broader collection of tools under the umbrella of the Distributed Machine Learning Community (DMLC)

The two key reasons for the usage of XGBoost as compared to other algorithms are model performance and execution speed. The algorithm was developed to efficiently reduce computing time, and allocate an optimal usage of memory resources.

Model Features:

- Gradient Boosting algorithm also called gradient boosting machine including the learning rate.
- Stochastic Gradient Boosting with sub-sampling at the row, column and column per split levels.
- Regularized Gradient Boosting with both L1 and L2 regularization.

System Features:

- Parallelization of tree construction using all CPU cores during training.
- Distributed Computing for training very large models using a cluster of machines.
- Out-of-Core Computing for very large datasets that don't fit into memory.
- Cache Optimization of data structures and algorithm to make best use of hardware.

Algorithm Features:

- Sparse Aware implementation with automatic handling of missing data values.
- Block Structure to support the parallelization of tree construction.
- Continued Training so that you can further boost an already fitted model on new data.

The XGBoost library implements the **gradient boosting decision tree** algorithm.

Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models, and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models. Gradient descent helps us minimize any differentiable function.

- An initial model F_0 is defined to predict the target variable y . This model will be associated with a residual $(y - F_0)$.
- A new model h_1 is fit to the residuals from the previous step.
- Now, F_0 and h_1 are combined to give F_1 , the boosted version of F_0 . The mean squared error from F_1 will be lower than that from F_0 :

$$F_1(x) <- F_0(x) + h_1(x)$$

- To improve the performance of F_1 , we could model after the residuals of F_1 and create a new model F_2 :

$$F_2(x) <- F_1(x) + h_2(x)$$

- This can be done for ' m ' iterations, until residuals have been minimized as much as possible:

$$F_m(x) <- F_{m-1}(x) + h_m(x)$$

CATBOOST

CatBoost stands for Categorical Boosting, and was developed by researchers at Yandex in 2017. It is widely used within the company for ranking tasks, forecasting and making recommendations.

CatBoost introduces two critical algorithmic advances:

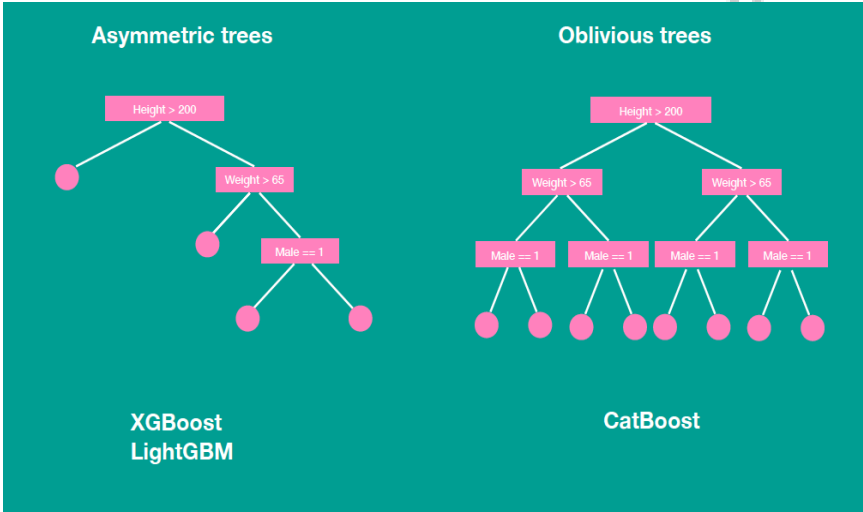
- The implementation of ordered boosting, a permutation-driven alternative to the classic algorithm: Gradients used at each step are estimated using the target values of the same data points the current model is built on. This leads to a shift of the distribution of estimated gradients in comparison with the true distribution of gradients, which leads to overfitting (prediction shift).
As a solution to prediction shift, CatBoost samples a new dataset independently at each step of boosting to obtain unshifted residuals by applying the current model to new training instances. CatBoost maintains a set of models differing by instances used for their training. Then, for calculating the residual on an instance, it uses a model trained without it. This is achieved using a random permutation σ of the training examples. Each model is learned using only the 1st i examples in the permutation. At each step, in order to obtain the residual for j^{th} sample, it uses M_{j-1} model. This is referred to as ordered boosting.
- An innovative algorithm for processing categorical features: It uses a method called Ordered Target Statistics.
 - Permuting the set of input observations in a random order. Multiple random permutations are generated.
 - All categorical feature values are transformed to numeric values using the following formula:

$$avg_target = \frac{countInClass + prior}{totalCount + 1}$$

Here, countInClass is the number of times the label value was equal to “1” for objects with the current categorical feature value, prior is the preliminary value for the numerator. It is determined by the starting parameters.
totalCount is the total number of objects (up to the current one) that have a categorical feature value matching the current one.

Some other features which make CatBoost a sought after algorithm are:

- It uses oblivious decision trees, where the same splitting criterion is used across an entire level of the tree. Such trees are balanced, less prone to overfitting, and allow speeding up prediction significantly at testing time.



- Feature Combinations: It uses combinations of categorical features as additional categorical features which capture high-order interactions. Consider that the objects in the training set belong to two categorical features: the musical genre (“rock”, “indie”) and the musical style (“dance”, “classical”). These features can occur in different combinations. CatBoost can create a new feature that is a combination of those listed (“dance rock”, “classic rock”, “dance indie”, or “indie classical”). Any number of features can be combined.
- Reduced time spent on parameter tuning, because it provides great results with default parameters.
- Fast and scalable GPU implementation.
- Faster predictions.