构建标准化建模代码（下）

C 课程目录
ONTENTS

1 变量分箱

2 变量评价指标计算和woe编码

3 变量筛选

4 模型训练

5 模型评估

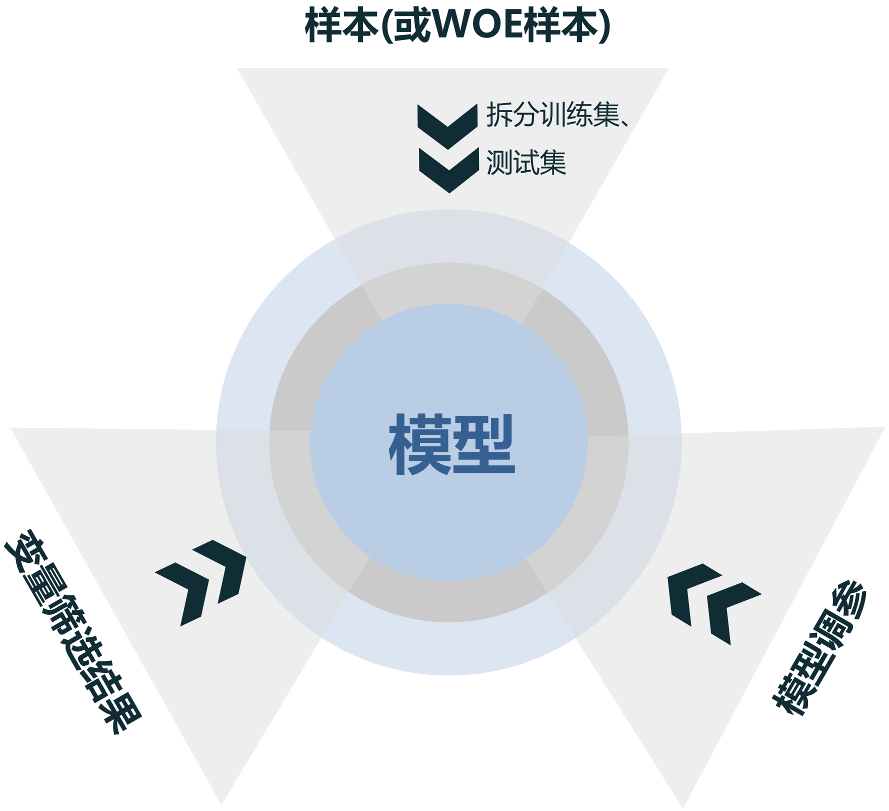
6 模型结果标准化输出

4

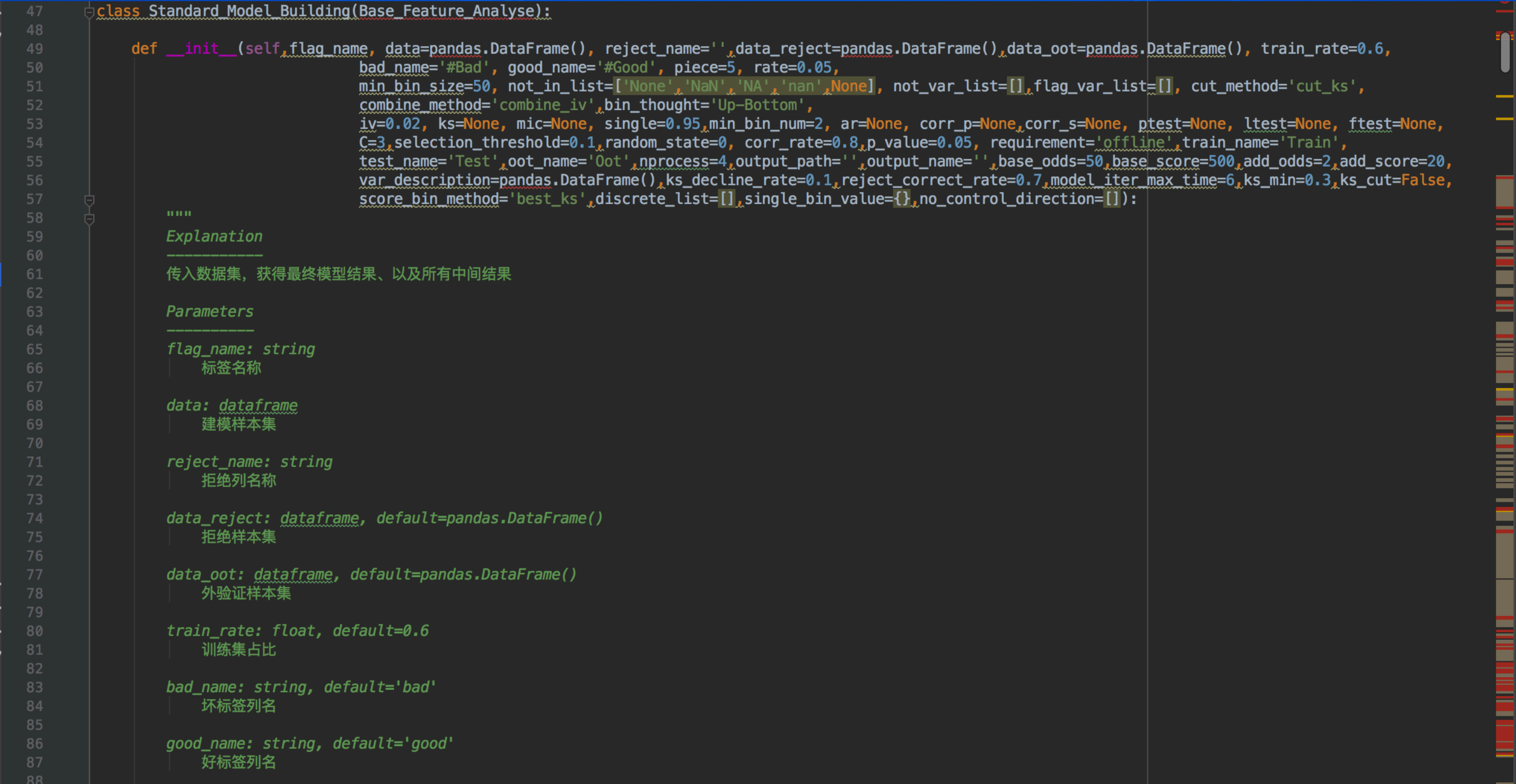
code demo

模型训练

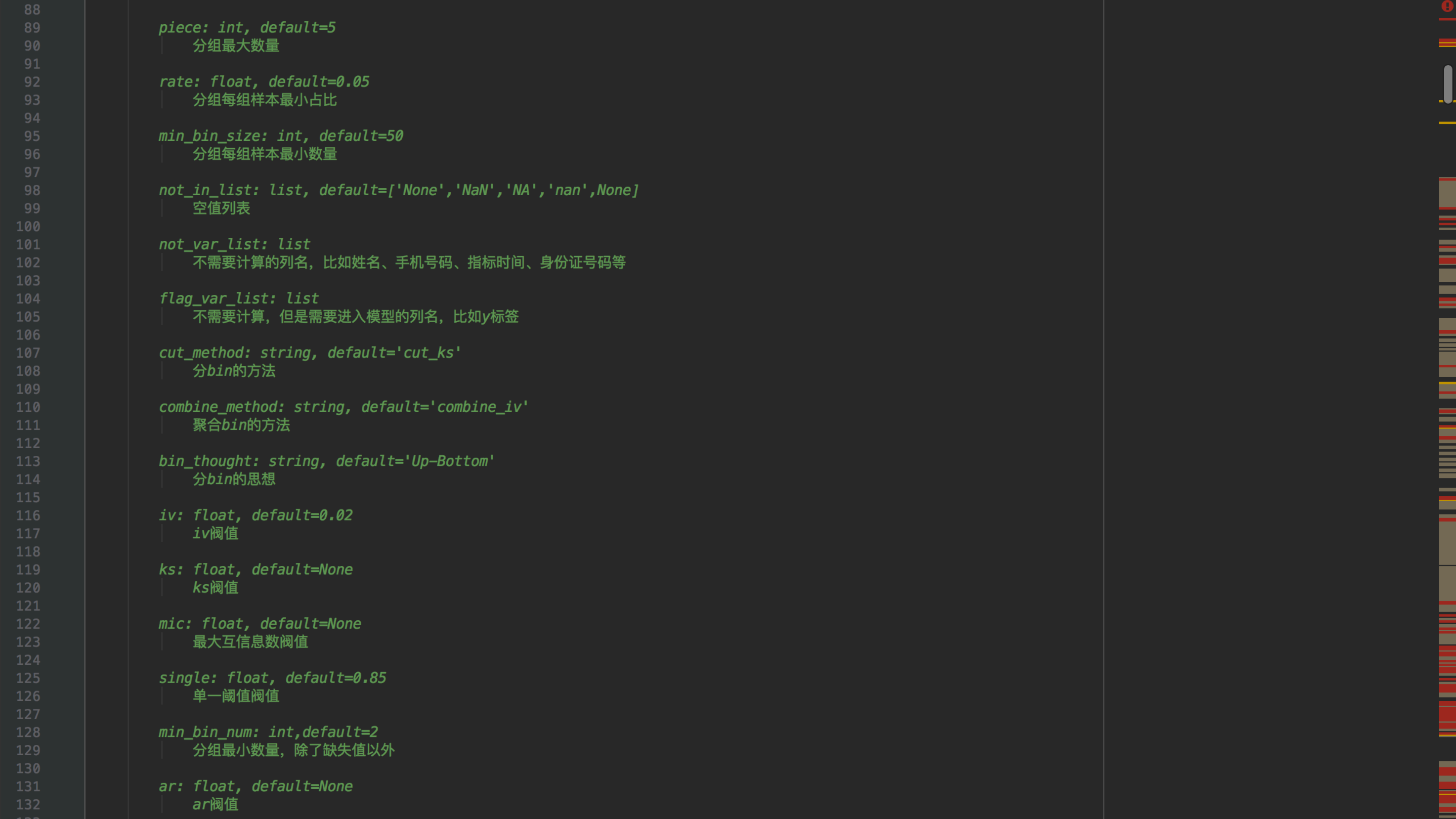
模型训练



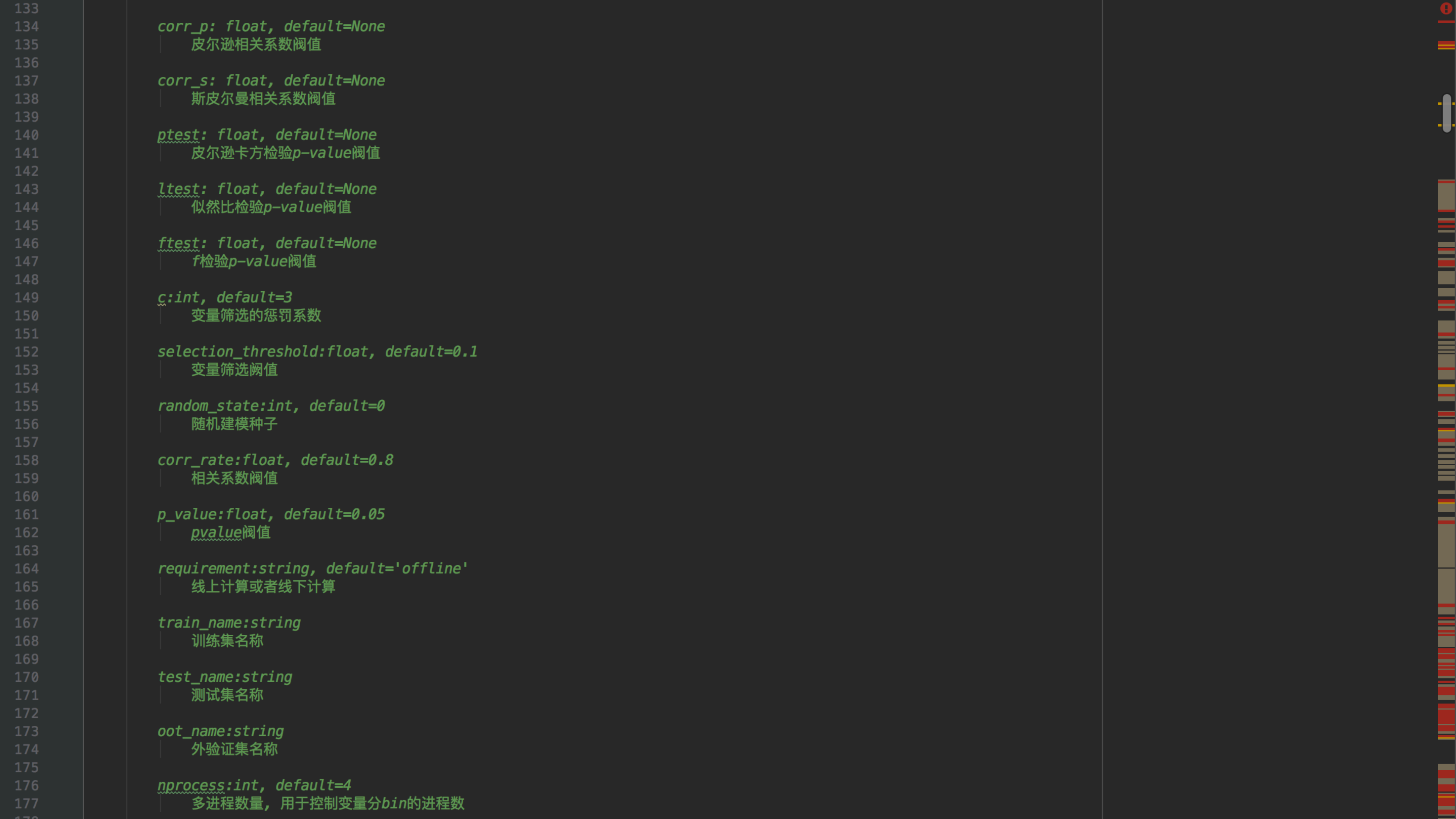
1 模型训练



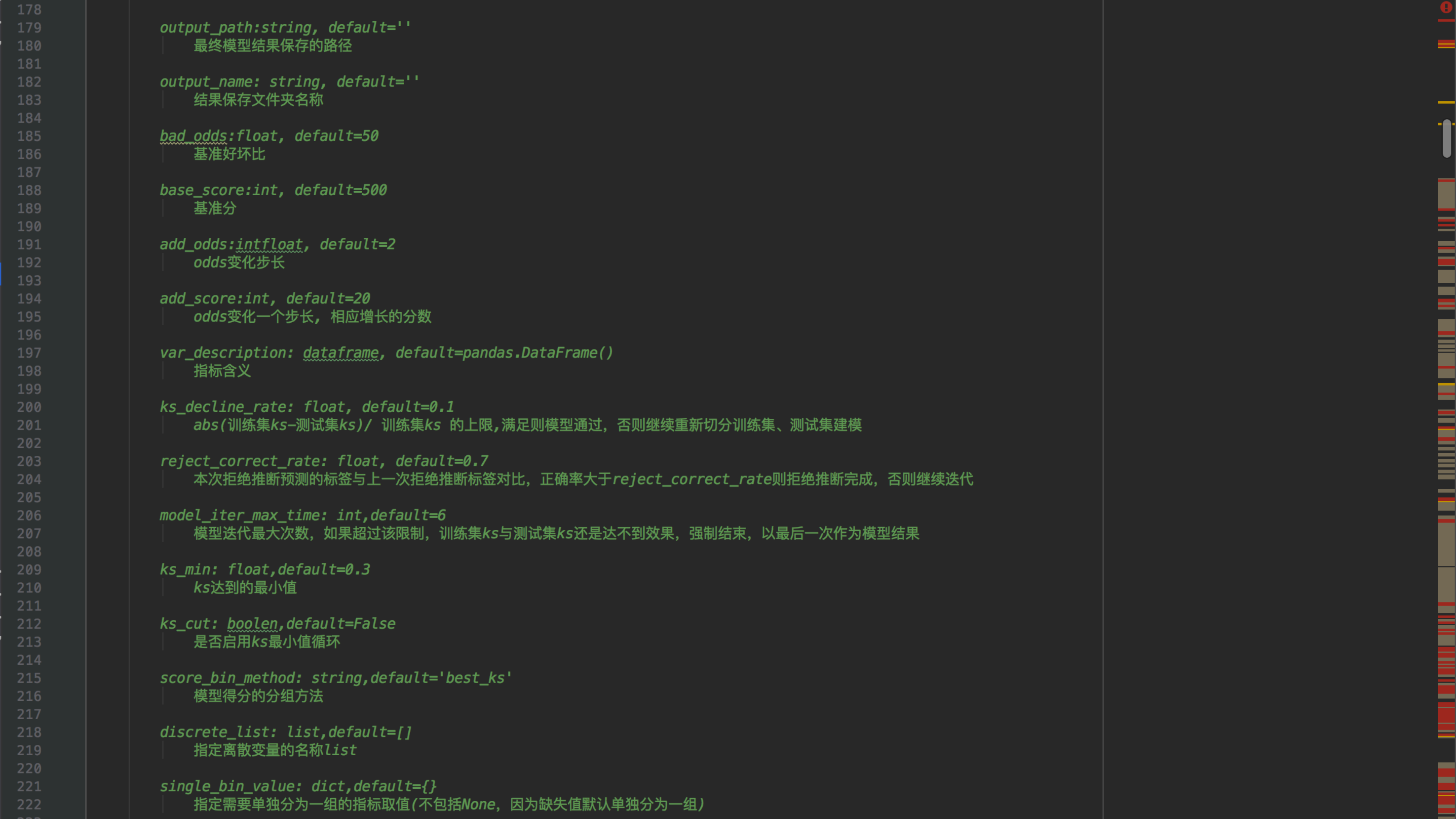
```python
class Standard_Model_Building(Base_Feature_Analyse):

    def __init__(self,flag_name, data=pandas.DataFrame(), reject_name='',data_reject=pandas.DataFrame(),data_oot=pandas.DataFrame(), train_rate=0.6,
                                 bad_name='#Bad', good_name='#Good', piece=5, rate=0.05,
                                 min_bin_size=50, not_in_list=['None','NaN','NA','nan',None], not_var_list=[],flag_var_list=[], cut_method='cut_ks',
                                 combine_method='combine_iv',bin_thought='Up-Bottom',
                                 iv=0.02, ks=None, mic=None, single=0.95,min_bin_num=2, ar=None, corr_p=None,corr_s=None, ptest=None, ltest=None, ftest=None,
                                 C=3,selection_threshold=0.1,random_state=0, corr_rate=0.8,p_value=0.05, requirement='offline',train_name='Train',
                                 test_name='Test',oot_name='Oot',nprocess=4,output_path='',output_name='',base_odds=50,base_score=500,add_odds=2,add_score=20,
                                 var_description=pandas.DataFrame(),ks_decline_rate=0.1,reject_correct_rate=0.7,model_iter_max_time=6,ks_min=0.3,ks_cut=False,
                                 score_bin_method='best_ks',discrete_list=[],single_bin_value={},no_control_direction=[]):
        """
        Explanation
        ------------
        传入数据集，获得最终模型结果、以及所有中间结果

        Parameters
        ------------
        flag_name: string
            标签名称

        data: dataframe
            建模样本集

        reject_name: string
            拒绝列名称

        data_reject: dataframe, default=pandas.DataFrame()
            拒绝样本集

        data_oot: dataframe, default=pandas.DataFrame()
            外验证样本集

        train_rate: float, default=0.6
            训练集占比

        bad_name: string, default='bad'
            坏标签列名

        good_name: string, default='good'
            好标签列名

```



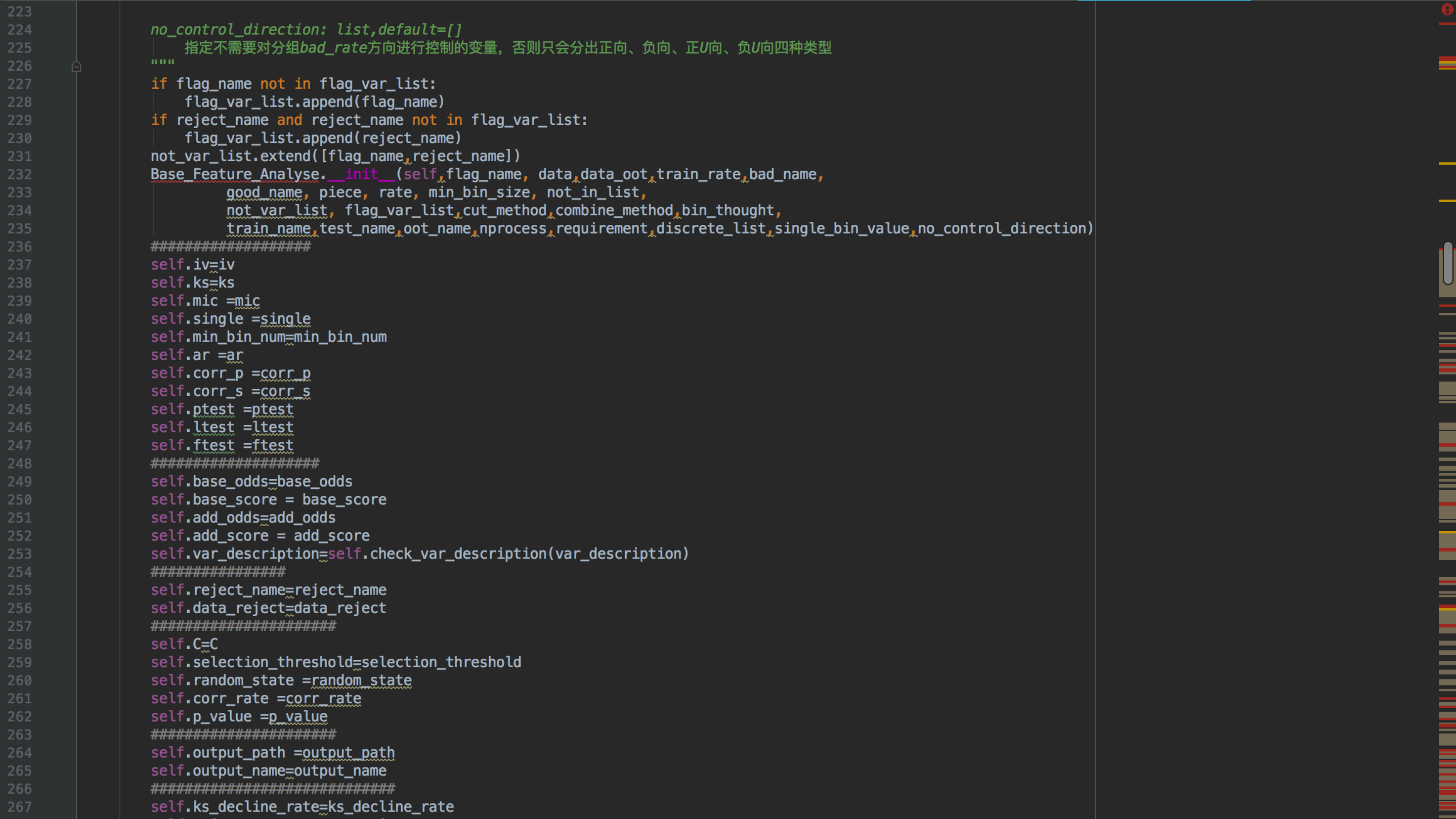
```

        piece: int, default=5
            分组最大数量

        rate: float, default=0.05
            分组每组样本最小占比

        min_bin_size: int, default=50
            分组每组样本最小数量

        not_in_list: list, default=['None','NaN','NA','nan',None]
            空值列表

        not_var_list: list
            不需要计算的列名，比如姓名、手机号码、指标时间、身份证号码等

        flag_var_list: list
            不需要计算，但是需要进入模型的列名，比如y标签

        cut_method: string, default='cut_ks'
            分bin的方法

        combine_method: string, default='combine_iv'
            聚合bin的方法

        bin_thought: string, default='Up-Bottom'
            分bin的思想

        iv: float, default=0.02
            iv阀值

        ks: float, default=None
            ks阀值

        mic: float, default=None
            最大互信息数阀值

        single: float, default=0.85
            单一阈值阀值

        min_bin_num: int,default=2
            分组最小数量，除了缺失值以外

        ar: float, default=None
            ar阀值
```



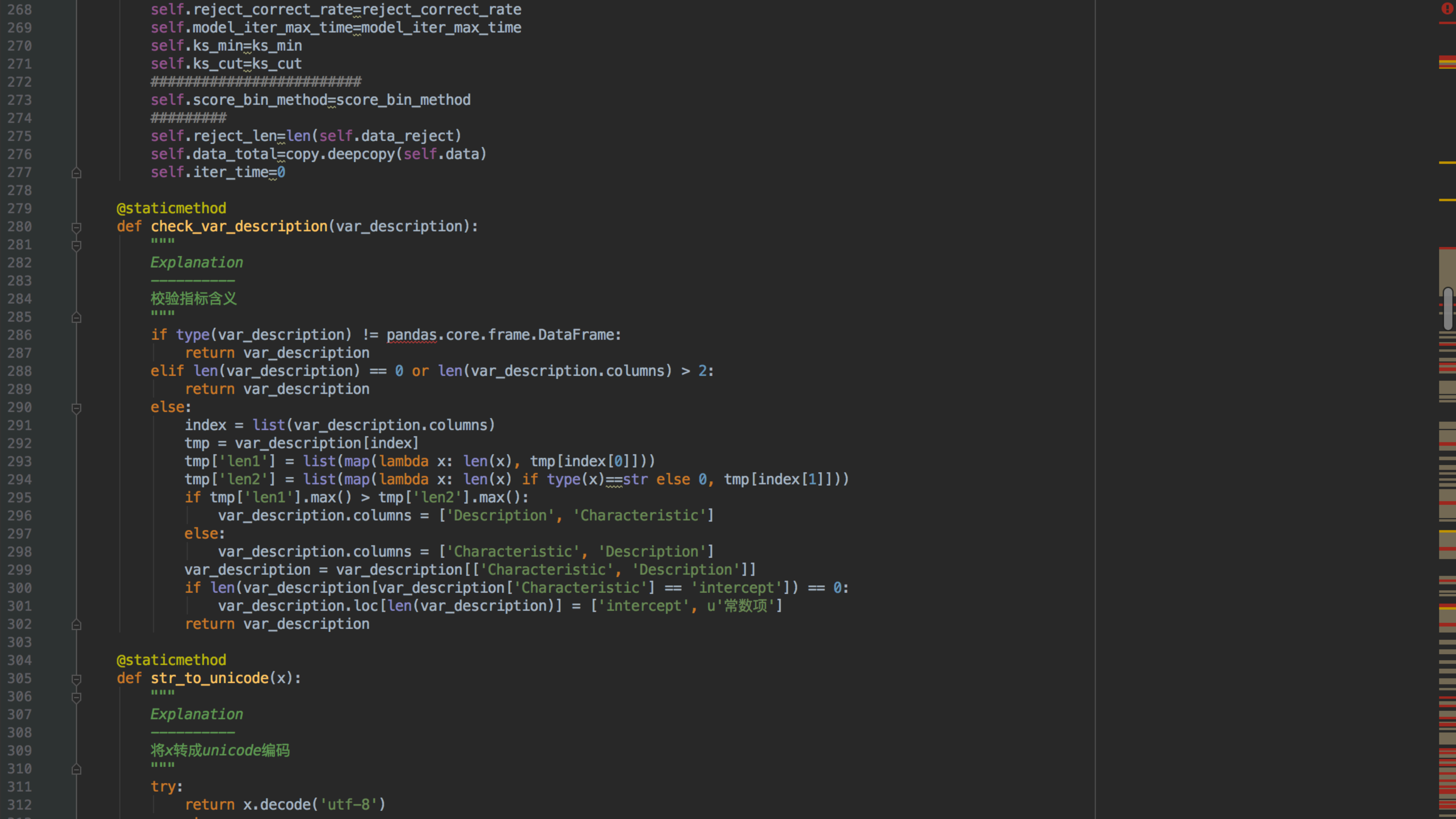
```
        corr_p: float, default=None
            皮尔逊相关系数阀值

        corr_s: float, default=None
            斯皮尔曼相关系数阀值

        ptest: float, default=None
            皮尔逊卡方检验p-value阀值

        ltest: float, default=None
            似然比检验p-value阀值

        ftest: float, default=None
            f检验p-value阀值

        c:int, default=3
            变量筛选的惩罚系数

        selection_threshold:float, default=0.1
            变量筛选阙值

        random_state:int, default=0
            随机建模种子

        corr_rate:float, default=0.8
            相关系数阀值

        p_value:float, default=0.05
            pvalue阀值

        requirement:string, default='offline'
            线上计算或者线下计算

        train_name:string
            训练集名称

        test_name:string
            测试集名称

        oot_name:string
            外验证集名称

        nprocess:int, default=4
            多进程数量，用于控制变量分bin的进程数
```



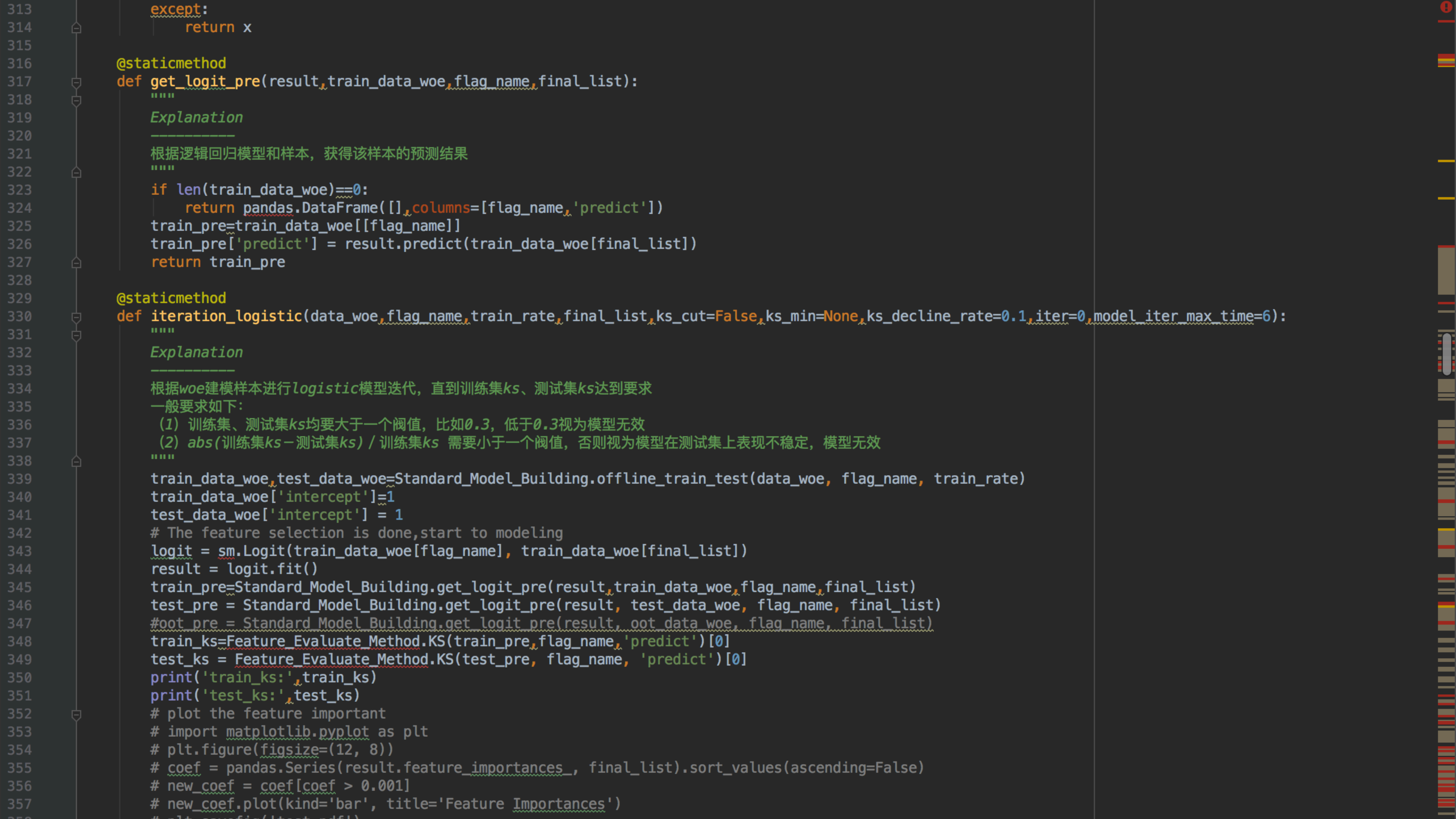
```

        output_path:string, default=''
            最终模型结果保存的路径

        output_name: string, default=''
            结果保存文件夹名称

        bad_odds:float, default=50
            基准好坏比

        base_score:int, default=500
            基准分

        add_odds:intfloat, default=2
            odds变化步长

        add_score:int, default=20
            odds变化一个步长，相应增长的分数

        var_description: dataframe, default=pandas.DataFrame()
            指标含义

        ks_decline_rate: float, default=0.1
            abs(训练集ks-测试集ks)/ 训练集ks 的上限,满足则模型通过，否则继续重新切分训练集、测试集建模

        reject_correct_rate: float, default=0.7
            本次拒绝推断预测的标签与上一次拒绝推断标签对比，正确率大于reject_correct_rate则拒绝推断完成，否则继续迭代

        model_iter_max_time: int,default=6
            模型迭代最大次数，如果超过该限制，训练集ks与测试集ks还是达不到效果，强制结束，以最后一次作为模型结果

        ks_min: float,default=0.3
            ks达到的最小值

        ks_cut: boolen,default=False
            是否启用ks最小值循环

        score_bin_method: string,default='best_ks'
            模型得分的分组方法

        discrete_list: list,default=[]
            指定离散变量的名称list

        single_bin_value: dict,default={}
            指定需要单独分为一组的指标取值(不包括None，因为缺失值默认单独分为一组)
```



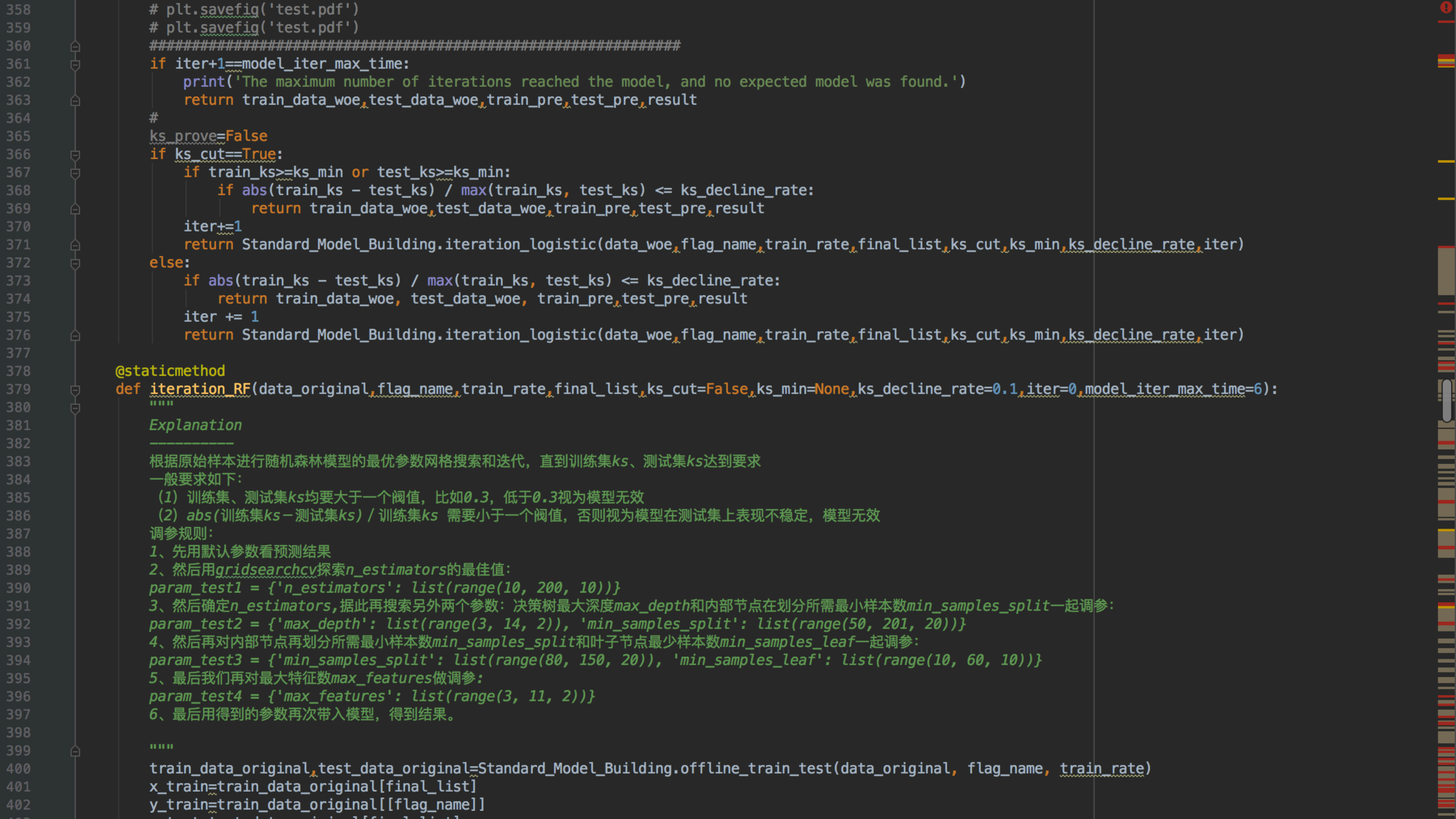
```python

        no_control_direction: list,default=[]
            指定不需要对分组bad_rate方向进行控制的变量，否则只会分出正向、负向、正U向、负U向四种类型
        """
        if flag_name not in flag_var_list:
            flag_var_list.append(flag_name)
        if reject_name and reject_name not in flag_var_list:
            flag_var_list.append(reject_name)
        not_var_list.extend([flag_name,reject_name])
        Base_Feature_Analyse.__init__(self,flag_name, data,data_oot,train_rate,bad_name,
                 good_name, piece, rate, min_bin_size, not_in_list,
                 not_var_list, flag_var_list,cut_method,combine_method,bin_thought,
                 train_name,test_name,oot_name,nprocess,requirement,discrete_list,single_bin_value,no_control_direction)
        ###################
        self.iv=iv
        self.ks=ks
        self.mic =mic
        self.single =single
        self.min_bin_num=min_bin_num
        self.ar =ar
        self.corr_p =corr_p
        self.corr_s =corr_s
        self.ptest =ptest
        self.ltest =ltest
        self.ftest =ftest
        ####################
        self.base_odds=base_odds
        self.base_score = base_score
        self.add_odds=add_odds
        self.add_score = add_score
        self.var_description=self.check_var_description(var_description)
        ################
        self.reject_name=reject_name
        self.data_reject=data_reject
        ######################
        self.C=C
        self.selection_threshold=selection_threshold
        self.random_state =random_state
        self.corr_rate =corr_rate
        self.p_value =p_value
        ######################
        self.output_path =output_path
        self.output_name=output_name
        #############################
        self.ks_decline_rate=ks_decline_rate
```



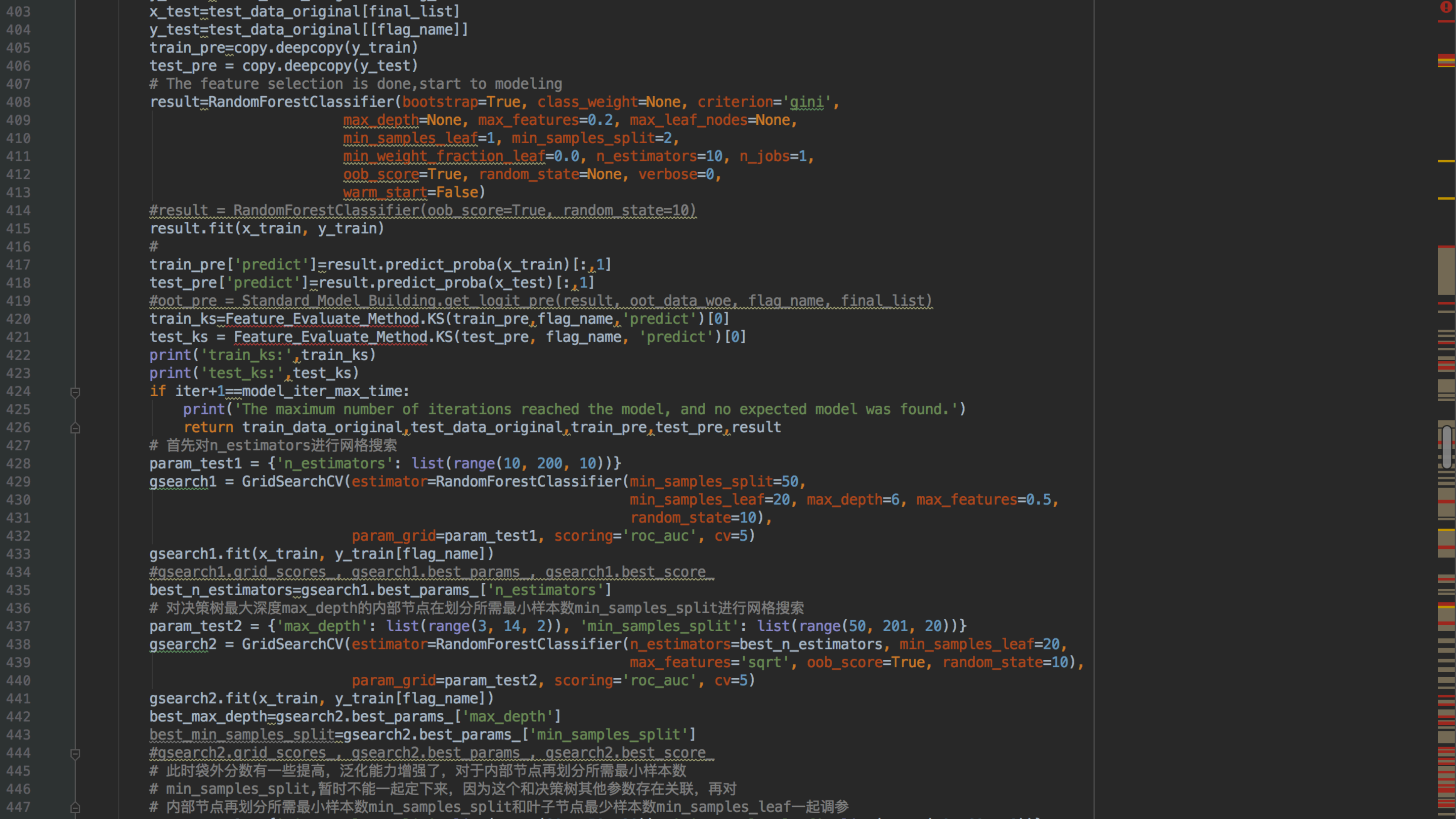
```
        self.reject_correct_rate=reject_correct_rate
        self.model_iter_max_time=model_iter_max_time
        self.ks_min=ks_min
        self.ks_cut=ks_cut
        #########################
        self.score_bin_method=score_bin_method
        #########
        self.reject_len=len(self.data_reject)
        self.data_total=copy.deepcopy(self.data)
        self.iter_time=0

    @staticmethod
    def check_var_description(var_description):
        """
        Explanation
        ----------
        校验指标含义
        """
        if type(var_description) != pandas.core.frame.DataFrame:
            return var_description
        elif len(var_description) == 0 or len(var_description.columns) > 2:
            return var_description
        else:
            index = list(var_description.columns)
            tmp = var_description[index]
            tmp['len1'] = list(map(lambda x: len(x), tmp[index[0]]))
            tmp['len2'] = list(map(lambda x: len(x) if type(x)==str else 0, tmp[index[1]]))
            if tmp['len1'].max() > tmp['len2'].max():
                var_description.columns = ['Description', 'Characteristic']
            else:
                var_description.columns = ['Characteristic', 'Description']
            var_description = var_description[['Characteristic', 'Description']]
            if len(var_description[var_description['Characteristic'] == 'intercept']) == 0:
                var_description.loc[len(var_description)] = ['intercept', u'常数项']
            return var_description

    @staticmethod
    def str_to_unicode(x):
        """
        Explanation
        ----------
        将x转成unicode编码
        """
        try:
            return x.decode('utf-8')
```



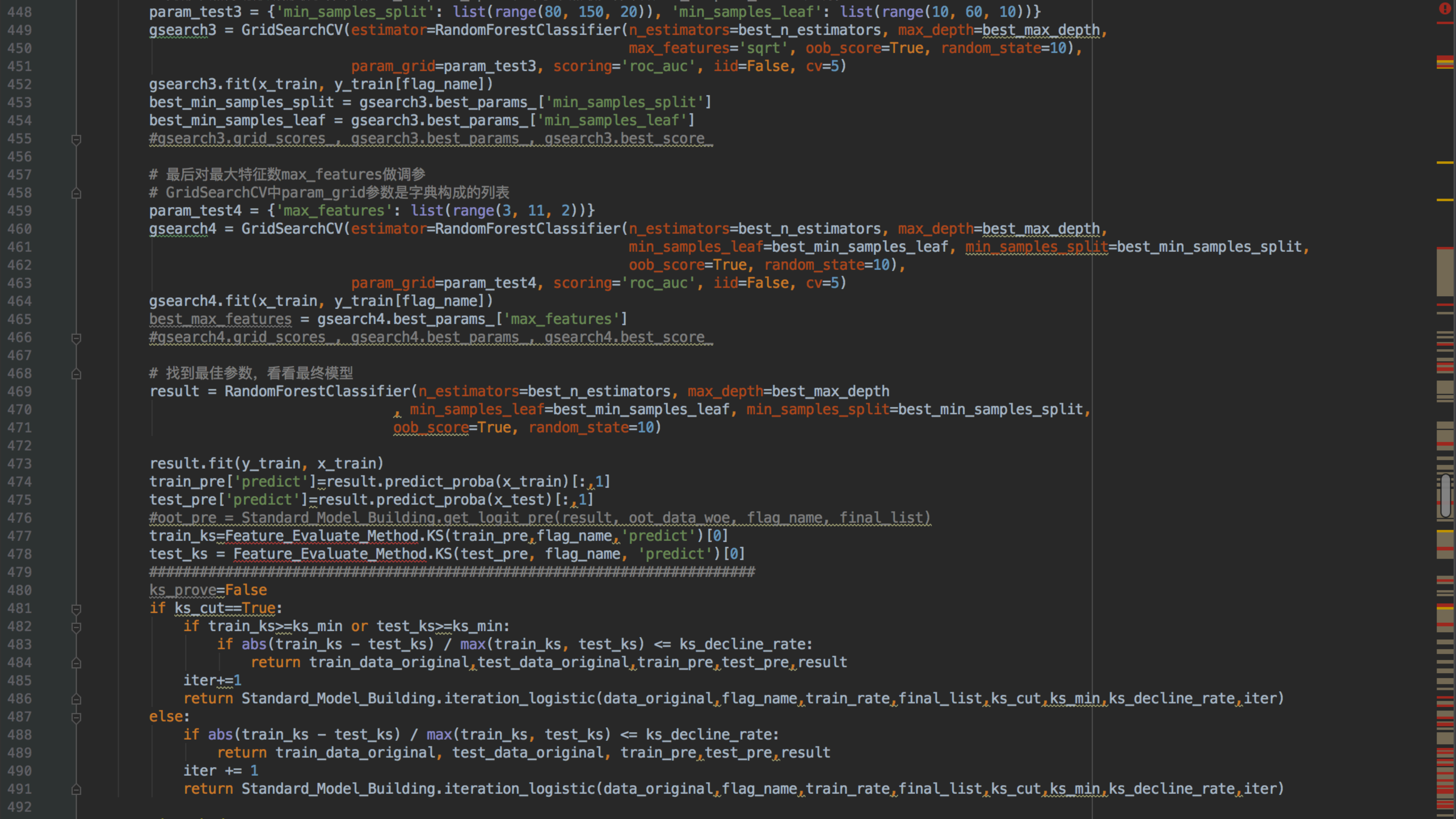
```
        except:
            return x

    @staticmethod
    def get_logit_pre(result,train_data_woe,flag_name,final_list):
        """
        Explanation
        ----------
        根据逻辑回归模型和样本，获得该样本的预测结果
        """
        if len(train_data_woe)==0:
            return pandas.DataFrame([],columns=[flag_name,'predict'])
        train_pre=train_data_woe[[flag_name]]
        train_pre['predict'] = result.predict(train_data_woe[final_list])
        return train_pre

    @staticmethod
    def iteration_logistic(data_woe,flag_name,train_rate,final_list,ks_cut=False,ks_min=None,ks_decline_rate=0.1,iter=0,model_iter_max_time=6):
        """
        Explanation
        ----------
        根据woe建模样本进行logistic模型迭代，直到训练集ks、测试集ks达到要求
        一般要求如下：
         (1) 训练集、测试集ks均要大于一个阀值，比如0.3，低于0.3视为模型无效
         (2) abs(训练集ks-测试集ks)/训练集ks 需要小于一个阀值，否则视为模型在测试集上表现不稳定，模型无效
        """
        train_data_woe,test_data_woe=Standard_Model_Building.offline_train_test(data_woe, flag_name, train_rate)
        train_data_woe['intercept']=1
        test_data_woe['intercept'] = 1
        # The feature selection is done,start to modeling
        logit = sm.Logit(train_data_woe[flag_name], train_data_woe[final_list])
        result = logit.fit()
        train_pre=Standard_Model_Building.get_logit_pre(result,train_data_woe,flag_name,final_list)
        test_pre = Standard_Model_Building.get_logit_pre(result, test_data_woe, flag_name, final_list)
        #oot_pre = Standard_Model_Building.get_logit_pre(result, oot_data_woe, flag_name, final_list)
        train_ks=Feature_Evaluate_Method.KS(train_pre,flag_name,'predict')[0]
        test_ks = Feature_Evaluate_Method.KS(test_pre, flag_name, 'predict')[0]
        print('train_ks:',train_ks)
        print('test_ks:',test_ks)
        # plot the feature important
        # import matplotlib.pyplot as plt
        # plt.figure(figsize=(12, 8))
        # coef = pandas.Series(result.feature_importances_, final_list).sort_values(ascending=False)
        # new_coef = coef[coef > 0.001]
        # new_coef.plot(kind='bar', title='Feature Importances')
        # plt.savefig('test.pdf')
```



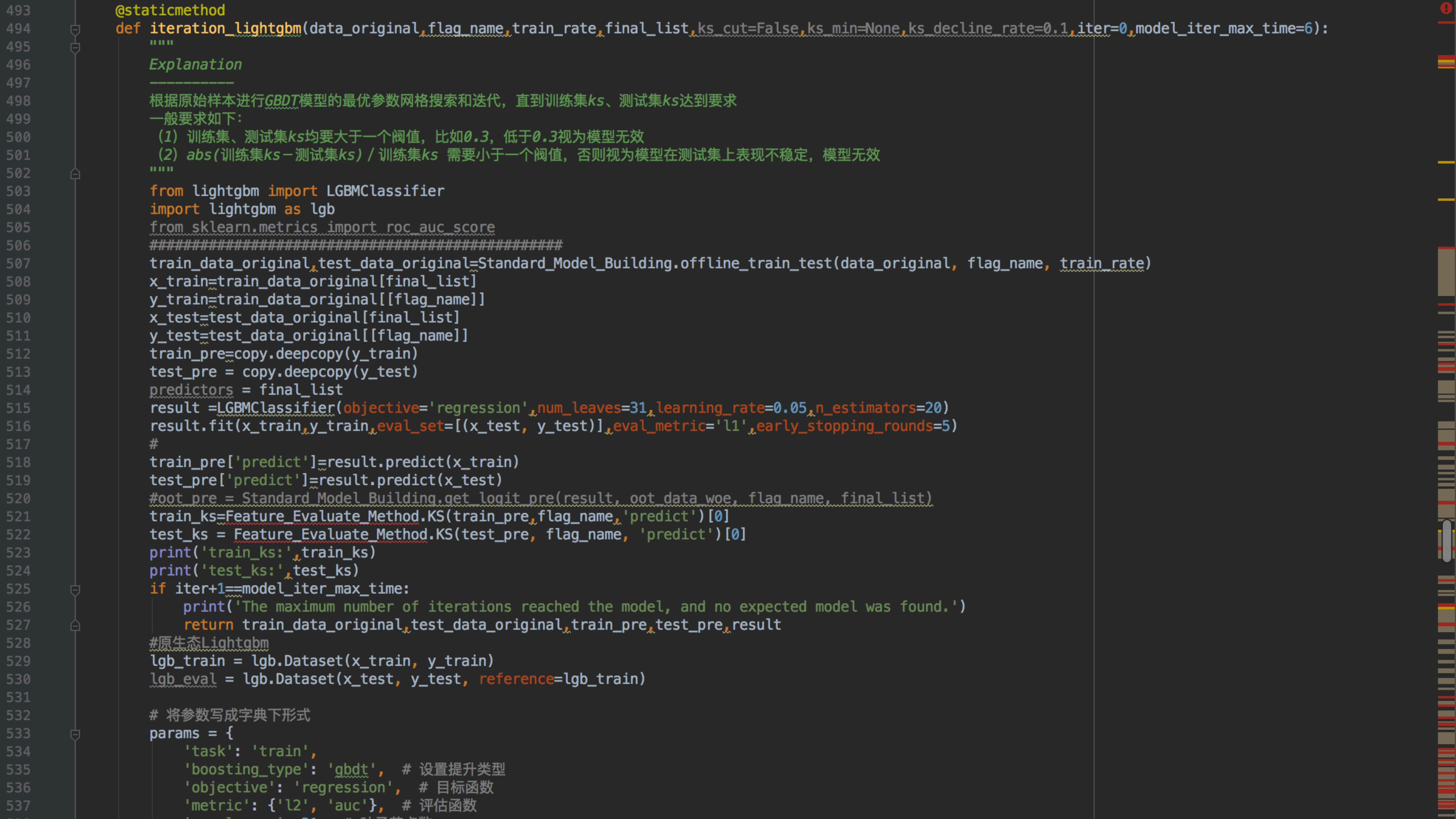
```
        # plt.savefig('test.pdf')
        # plt.savefig('test.pdf')
        ###############################################################
        if iter+1==model_iter_max_time:
            print('The maximum number of iterations reached the model, and no expected model was found.')
            return train_data_woe,test_data_woe,train_pre,test_pre,result
        #
        ks_prove=False
        if ks_cut==True:
            if train_ks>=ks_min or test_ks>=ks_min:
                if abs(train_ks - test_ks) / max(train_ks, test_ks) <= ks_decline_rate:
                    return train_data_woe,test_data_woe,train_pre,test_pre,result
            iter+=1
            return Standard_Model_Building.iteration_logistic(data_woe,flag_name,train_rate,final_list,ks_cut,ks_min,ks_decline_rate,iter)
        else:
            if abs(train_ks - test_ks) / max(train_ks, test_ks) <= ks_decline_rate:
                return train_data_woe, test_data_woe, train_pre,test_pre,result
            iter += 1
            return Standard_Model_Building.iteration_logistic(data_woe,flag_name,train_rate,final_list,ks_cut,ks_min,ks_decline_rate,iter)

    @staticmethod
    def iteration_RF(data_original,flag_name,train_rate,final_list,ks_cut=False,ks_min=None,ks_decline_rate=0.1,iter=0,model_iter_max_time=6):
        """
        Explanation
        ----------
        根据原始样本进行随机森林模型的最优参数网格搜索和迭代，直到训练集ks、测试集ks达到要求
        一般要求如下：
        （1）训练集、测试集ks均要大于一个阈值，比如0.3，低于0.3视为模型无效
        （2）abs(训练集ks-测试集ks) / 训练集ks 需要小于一个阈值，否则视为模型在测试集上表现不稳定，模型无效
        调参规则：
        1、先用默认参数看预测结果
        2、然后用gridsearchcv探索n_estimators的最佳值：
        param_test1 = {'n_estimators': list(range(10, 200, 10))}
        3、然后确定n_estimators,据此再搜索另外两个参数：决策树最大深度max_depth和内部节点在划分所需最小样本数min_samples_split一起调参：
        param_test2 = {'max_depth': list(range(3, 14, 2)), 'min_samples_split': list(range(50, 201, 20))}
        4、然后再对内部节点再划分所需最小样本数min_samples_split和叶子节点最少样本数min_samples_leaf一起调参：
        param_test3 = {'min_samples_split': list(range(80, 150, 20)), 'min_samples_leaf': list(range(10, 60, 10))}
        5、最后我们再对最大特征数max_features做调参：
        param_test4 = {'max_features': list(range(3, 11, 2))}
        6、最后用得到的参数再次带入模型，得到结果。

        """
        train_data_original,test_data_original=Standard_Model_Building.offline_train_test(data_original, flag_name, train_rate)
        x_train=train_data_original[final_list]
        y_train=train_data_original[[flag_name]]
```



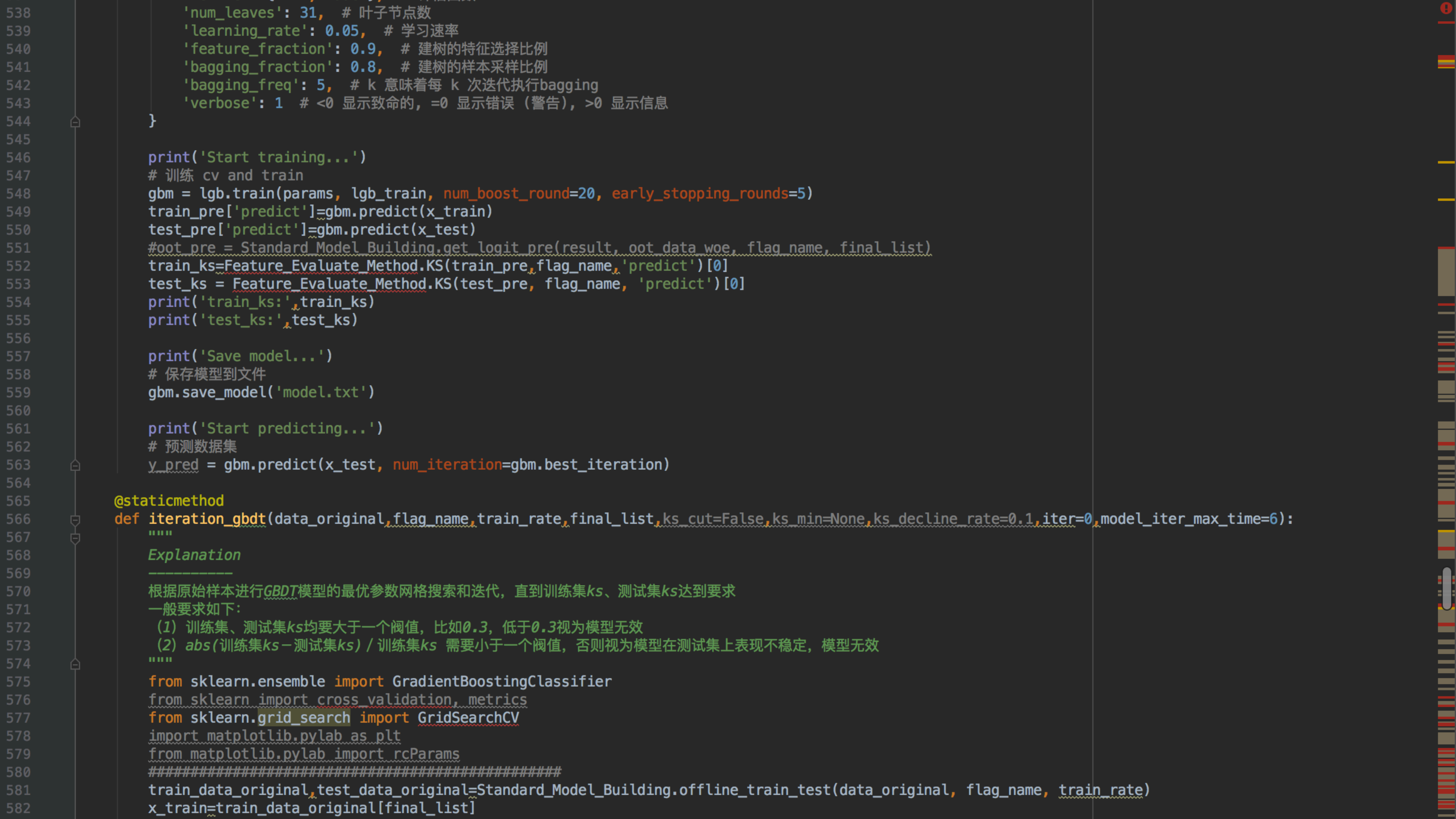
```
        x_test=test_data_original[final_list]
        y_test=test_data_original[[flag_name]]
        train_pre=copy.deepcopy(y_train)
        test_pre = copy.deepcopy(y_test)
        # The feature selection is done,start to modeling
        result=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                               max_depth=None, max_features=0.2, max_leaf_nodes=None,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                               oob_score=True, random_state=None, verbose=0,
                               warm_start=False)
        #result = RandomForestClassifier(oob_score=True, random_state=10)
        result.fit(x_train, y_train)
        #
        train_pre['predict']=result.predict_proba(x_train)[:,1]
        test_pre['predict']=result.predict_proba(x_test)[:,1]
        #oot_pre = Standard_Model_Building.get_logit_pre(result, oot_data_woe, flag_name, final_list)
        train_ks=Feature_Evaluate_Method.KS(train_pre,flag_name,'predict')[0]
        test_ks = Feature_Evaluate_Method.KS(test_pre, flag_name, 'predict')[0]
        print('train_ks:',train_ks)
        print('test_ks:',test_ks)
        if iter+1==model_iter_max_time:
            print('The maximum number of iterations reached the model, and no expected model was found.')
            return train_data_original,test_data_original,train_pre,test_pre,result
        # 首先对n_estimators进行网格搜索
        param_test1 = {'n_estimators': list(range(10, 200, 10))}
        gsearch1 = GridSearchCV(estimator=RandomForestClassifier(min_samples_split=50,
                                                                 min_samples_leaf=20, max_depth=6, max_features=0.5,
                                                                 random_state=10),
                                param_grid=param_test1, scoring='roc_auc', cv=5)
        gsearch1.fit(x_train, y_train[flag_name])
        #gsearch1.grid_scores_, gsearch1.best_params_, gsearch1.best_score_
        best_n_estimators=gsearch1.best_params_['n_estimators']
        # 对决策树最大深度max_depth的内部节点在划分所需最小样本数min_samples_split进行网格搜索
        param_test2 = {'max_depth': list(range(3, 14, 2)), 'min_samples_split': list(range(50, 201, 20))}
        gsearch2 = GridSearchCV(estimator=RandomForestClassifier(n_estimators=best_n_estimators, min_samples_leaf=20,
                                                                 max_features='sqrt', oob_score=True, random_state=10),
                                param_grid=param_test2, scoring='roc_auc', cv=5)
        gsearch2.fit(x_train, y_train[flag_name])
        best_max_depth=gsearch2.best_params_['max_depth']
        best_min_samples_split=gsearch2.best_params_['min_samples_split']
        #gsearch2.grid_scores_, gsearch2.best_params_, gsearch2.best_score_
        # 此时袋外分数有一些提高，泛化能力增强了，对于内部节点再划分所需最小样本数
        # min_samples_split,暂时不能一起定下来，因为这个和决策树其他参数存在关联，再对
        # 内部节点再划分所需最小样本数min_samples_split和叶子节点最少样本数min_samples_leaf一起调参
```



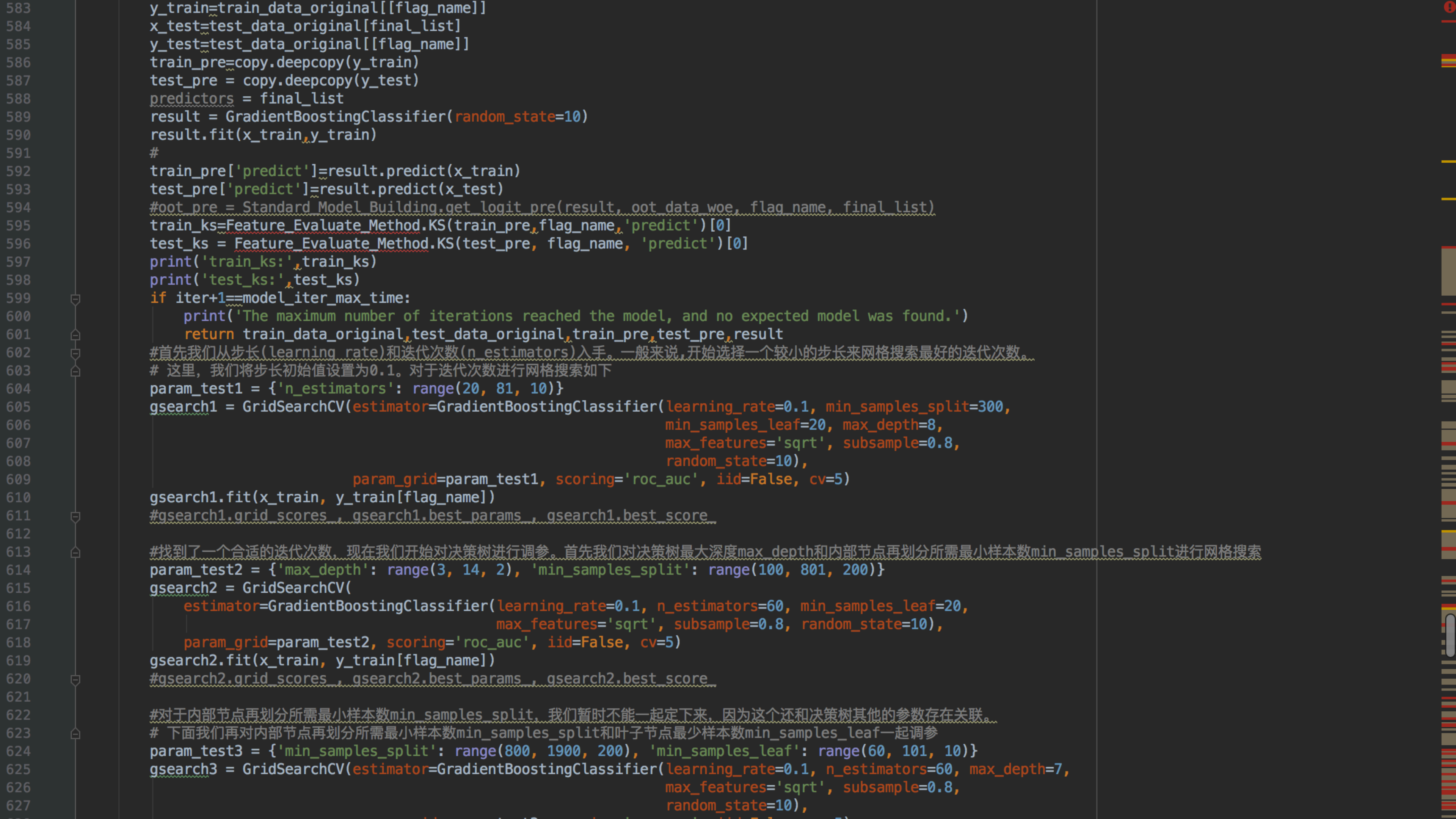
```python
    param_test3 = {'min_samples_split': list(range(80, 150, 20)), 'min_samples_leaf': list(range(10, 60, 10))}
    gsearch3 = GridSearchCV(estimator=RandomForestClassifier(n_estimators=best_n_estimators, max_depth=best_max_depth,
                                                             max_features='sqrt', oob_score=True, random_state=10),
                            param_grid=param_test3, scoring='roc_auc', iid=False, cv=5)
    gsearch3.fit(x_train, y_train[flag_name])
    best_min_samples_split = gsearch3.best_params_['min_samples_split']
    best_min_samples_leaf = gsearch3.best_params_['min_samples_leaf']
    #gsearch3.grid_scores_, gsearch3.best_params_, gsearch3.best_score_

    # 最后对最大特征数max_features做调参
    # GridSearchCV中param_grid参数是字典构成的列表
    param_test4 = {'max_features': list(range(3, 11, 2))}
    gsearch4 = GridSearchCV(estimator=RandomForestClassifier(n_estimators=best_n_estimators, max_depth=best_max_depth,
                                                             min_samples_leaf=best_min_samples_leaf, min_samples_split=best_min_samples_split,
                                                             oob_score=True, random_state=10),
                            param_grid=param_test4, scoring='roc_auc', iid=False, cv=5)
    gsearch4.fit(x_train, y_train[flag_name])
    best_max_features = gsearch4.best_params_['max_features']
    #gsearch4.grid_scores_, gsearch4.best_params_, gsearch4.best_score_

    # 找到最佳参数，看看最终模型
    result = RandomForestClassifier(n_estimators=best_n_estimators, max_depth=best_max_depth
                                 , min_samples_leaf=best_min_samples_leaf, min_samples_split=best_min_samples_split,
                                 oob_score=True, random_state=10)

    result.fit(y_train, x_train)
    train_pre['predict']=result.predict_proba(x_train)[:,1]
    test_pre['predict']=result.predict_proba(x_test)[:,1]
    #oot_pre = Standard_Model_Building.get_logit_pre(result, oot_data_woe, flag_name, final_list)
    train_ks=Feature_Evaluate_Method.KS(train_pre,flag_name,'predict')[0]
    test_ks = Feature_Evaluate_Method.KS(test_pre, flag_name, 'predict')[0]
    ########################################################################
    ks_prove=False
    if ks_cut==True:
        if train_ks>=ks_min or test_ks>=ks_min:
            if abs(train_ks - test_ks) / max(train_ks, test_ks) <= ks_decline_rate:
                return train_data_original,test_data_original,train_pre,test_pre,result
        iter+=1
        return Standard_Model_Building.iteration_logistic(data_original,flag_name,train_rate,final_list,ks_cut,ks_min,ks_decline_rate,iter)
    else:
        if abs(train_ks - test_ks) / max(train_ks, test_ks) <= ks_decline_rate:
            return train_data_original, test_data_original, train_pre,test_pre,result
        iter += 1
        return Standard_Model_Building.iteration_logistic(data_original,flag_name,train_rate,final_list,ks_cut,ks_min,ks_decline_rate,iter)

```



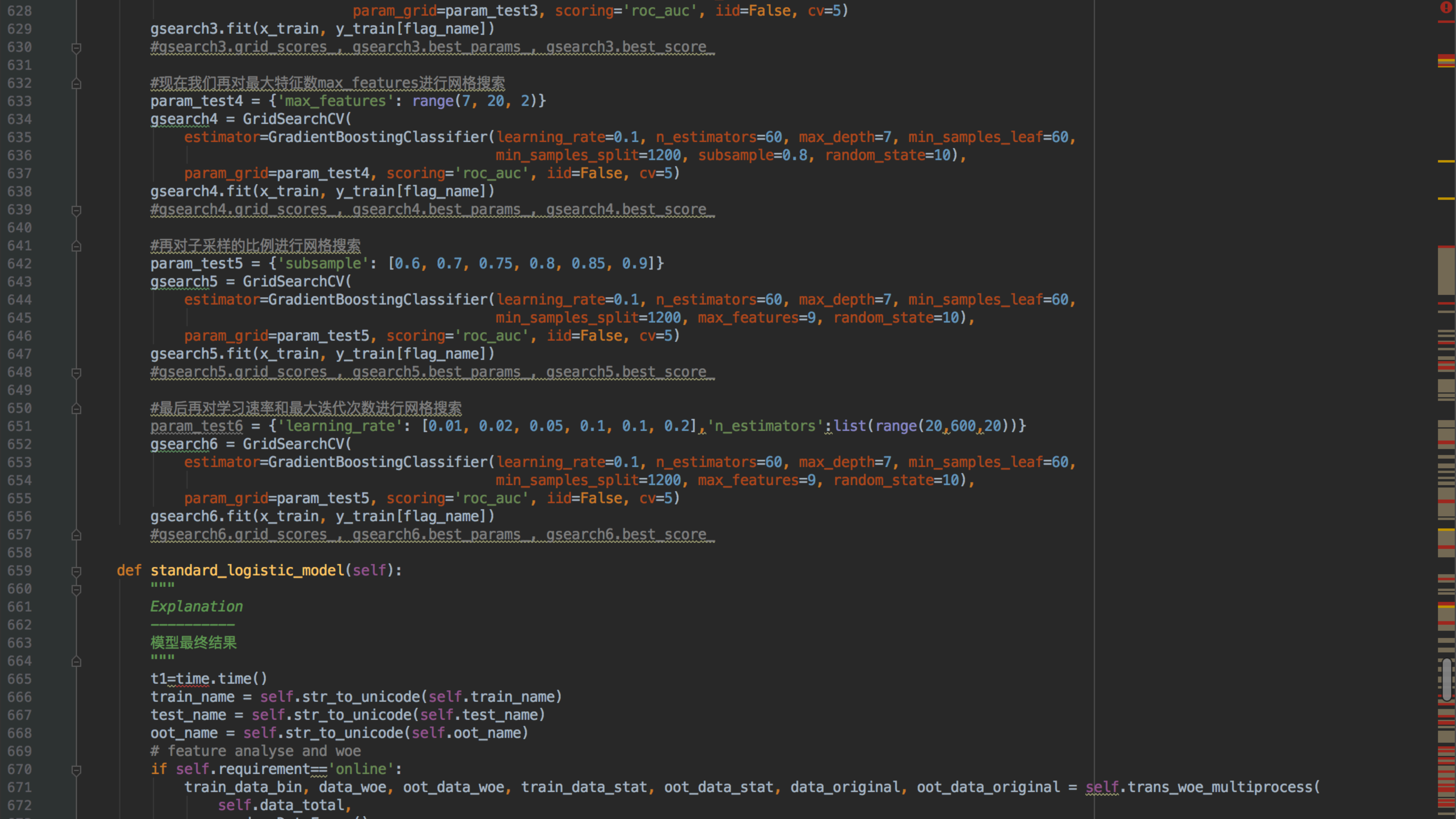
```python
    @staticmethod
    def iteration_lightgbm(data_original,flag_name,train_rate,final_list,ks_cut=False,ks_min=None,ks_decline_rate=0.1,iter=0,model_iter_max_time=6):
        """
        Explanation
        ------------
        根据原始样本进行GBDT模型的最优参数网格搜索和迭代，直到训练集ks、测试集ks达到要求
        一般要求如下：
         (1) 训练集、测试集ks均要大于一个阈值，比如0.3，低于0.3视为模型无效
         (2) abs(训练集ks－测试集ks) / 训练集ks 需要小于一个阈值，否则视为模型在测试集上表现不稳定，模型无效
        """
        from lightgbm import LGBMClassifier
        import lightgbm as lgb
        from sklearn.metrics import roc_auc_score
        ##################################################
        train_data_original,test_data_original=Standard_Model_Building.offline_train_test(data_original, flag_name, train_rate)
        x_train=train_data_original[final_list]
        y_train=train_data_original[[flag_name]]
        x_test=test_data_original[final_list]
        y_test=test_data_original[[flag_name]]
        train_pre=copy.deepcopy(y_train)
        test_pre = copy.deepcopy(y_test)
        predictors = final_list
        result =LGBMClassifier(objective='regression',num_leaves=31,learning_rate=0.05,n_estimators=20)
        result.fit(x_train,y_train,eval_set=[(x_test, y_test)],eval_metric='l1',early_stopping_rounds=5)
        #
        train_pre['predict']=result.predict(x_train)
        test_pre['predict']=result.predict(x_test)
        #oot_pre = Standard_Model_Building.get_logit_pre(result, oot_data_woe, flag_name, final_list)
        train_ks=Feature_Evaluate_Method.KS(train_pre,flag_name,'predict')[0]
        test_ks = Feature_Evaluate_Method.KS(test_pre, flag_name, 'predict')[0]
        print('train_ks:',train_ks)
        print('test_ks:',test_ks)
        if iter+1==model_iter_max_time:
            print('The maximum number of iterations reached the model, and no expected model was found.')
            return train_data_original,test_data_original,train_pre,test_pre,result
        #原生态Lightgbm
        lgb_train = lgb.Dataset(x_train, y_train)
        lgb_eval = lgb.Dataset(x_test, y_test, reference=lgb_train)

        # 将参数写成字典下形式
        params = {
            'task': 'train',
            'boosting_type': 'gbdt',  # 设置提升类型
            'objective': 'regression',  # 目标函数
            'metric': {'l2', 'auc'},  # 评估函数
```



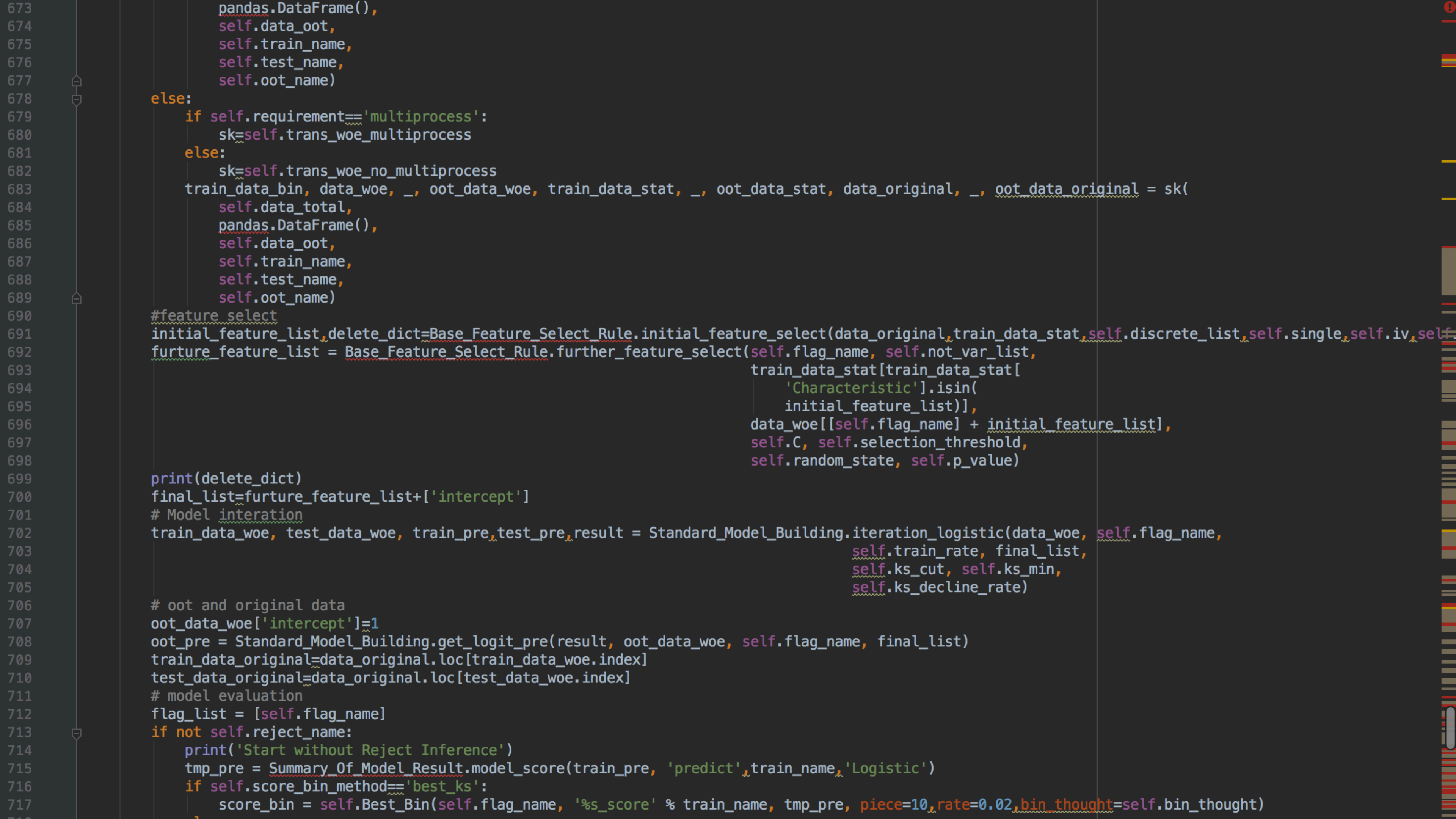
```
            'num_leaves': 31,  # 叶子节点数
            'learning_rate': 0.05,  # 学习速率
            'feature_fraction': 0.9,  # 建树的特征选择比例
            'bagging_fraction': 0.8,  # 建树的样本采样比例
            'bagging_freq': 5,  # k 意味着每 k 次迭代执行bagging
            'verbose': 1  # <0 显示致命的, =0 显示错误 (警告), >0 显示信息
        }

        print('Start training...')
        # 训练 cv and train
        gbm = lgb.train(params, lgb_train, num_boost_round=20, early_stopping_rounds=5)
        train_pre['predict']=gbm.predict(x_train)
        test_pre['predict']=gbm.predict(x_test)
        #oot_pre = Standard_Model_Building.get_logit_pre(result, oot_data_woe, flag_name, final_list)
        train_ks=Feature_Evaluate_Method.KS(train_pre,flag_name,'predict')[0]
        test_ks = Feature_Evaluate_Method.KS(test_pre, flag_name, 'predict')[0]
        print('train_ks:',train_ks)
        print('test_ks:',test_ks)

        print('Save model...')
        # 保存模型到文件
        gbm.save_model('model.txt')

        print('Start predicting...')
        # 预测数据集
        y_pred = gbm.predict(x_test, num_iteration=gbm.best_iteration)

    @staticmethod
    def iteration_gbdt(data_original,flag_name,train_rate,final_list,ks_cut=False,ks_min=None,ks_decline_rate=0.1,iter=0,model_iter_max_time=6):
        """
        Explanation
        ----------
        根据原始样本进行GBDT模型的最优参数网格搜索和迭代，直到训练集ks、测试集ks达到要求
        一般要求如下：
        （1）训练集、测试集ks均要大于一个阈值，比如0.3，低于0.3视为模型无效
        （2）abs(训练集ks-测试集ks)/训练集ks 需要小于一个阈值，否则视为模型在测试集上表现不稳定，模型无效
        """
        from sklearn.ensemble import GradientBoostingClassifier
        from sklearn import cross_validation, metrics
        from sklearn.grid_search import GridSearchCV
        import matplotlib.pylab as plt
        from matplotlib.pylab import rcParams
        ##################################################
        train_data_original,test_data_original=Standard_Model_Building.offline_train_test(data_original, flag_name, train_rate)
        x_train=train_data_original[final_list]
```



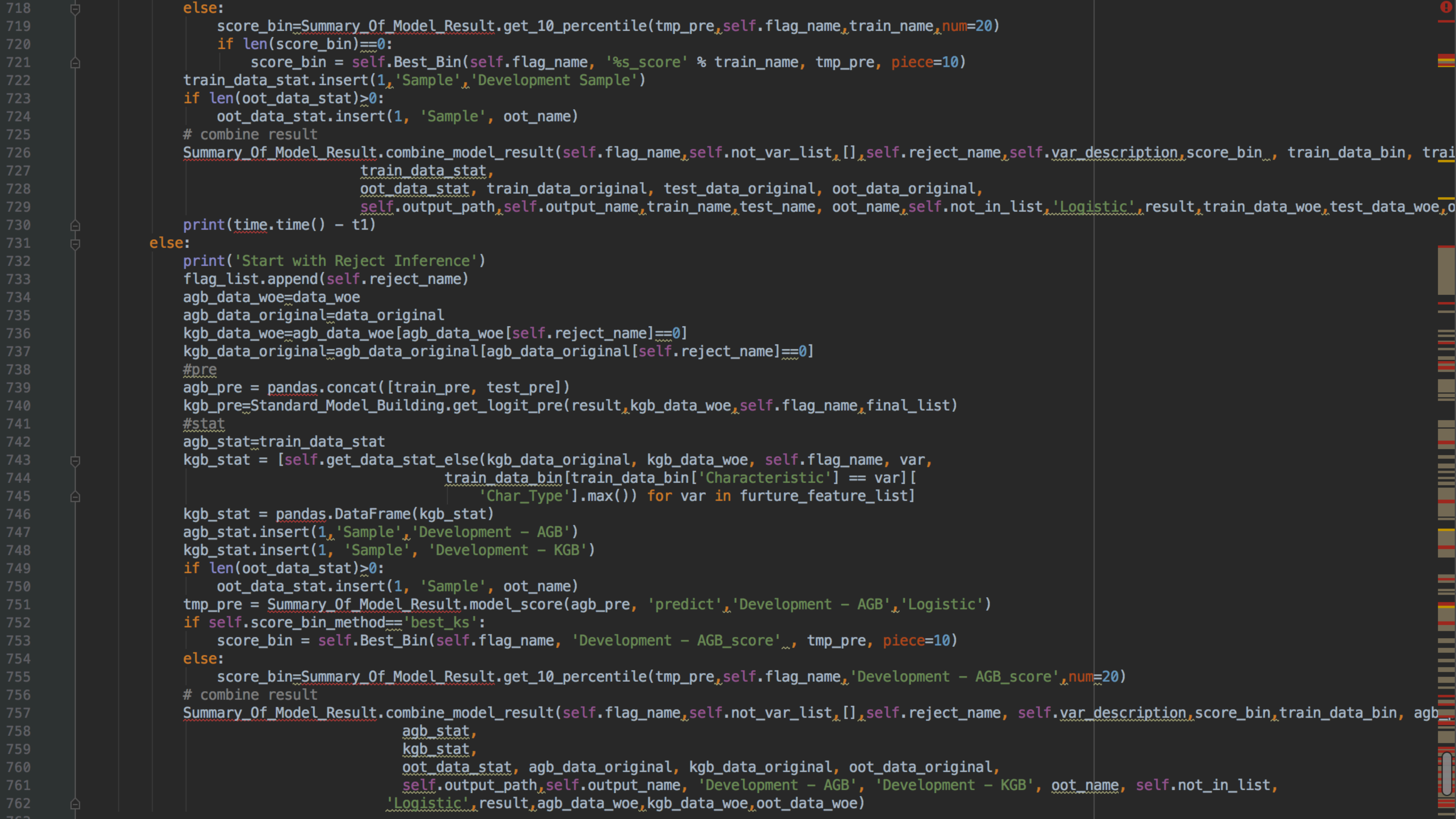
```
        y_train=train_data_original[[flag_name]]
        x_test=test_data_original[final_list]
        y_test=test_data_original[[flag_name]]
        train_pre=copy.deepcopy(y_train)
        test_pre = copy.deepcopy(y_test)
        predictors = final_list
        result = GradientBoostingClassifier(random_state=10)
        result.fit(x_train,y_train)
        #
        train_pre['predict']=result.predict(x_train)
        test_pre['predict']=result.predict(x_test)
        #oot_pre = Standard_Model_Building.get_logit_pre(result, oot_data_woe, flag_name, final_list)
        train_ks=Feature_Evaluate_Method.KS(train_pre,flag_name,'predict')[0]
        test_ks = Feature_Evaluate_Method.KS(test_pre, flag_name, 'predict')[0]
        print('train_ks:',train_ks)
        print('test_ks:',test_ks)
        if iter+1==model_iter_max_time:
            print('The maximum number of iterations reached the model, and no expected model was found.')
            return train_data_original,test_data_original,train_pre,test_pre,result
        #首先我们从步长(learning rate)和迭代次数(n_estimators)入手。一般来说,开始选择一个较小的步长来网格搜索最好的迭代次数。
        # 这里，我们将步长初始值设置为0.1。对于迭代次数进行网格搜索如下
        param_test1 = {'n_estimators': range(20, 81, 10)}
        gsearch1 = GridSearchCV(estimator=GradientBoostingClassifier(learning_rate=0.1, min_samples_split=300,
                                                                     min_samples_leaf=20, max_depth=8,
                                                                     max_features='sqrt', subsample=0.8,
                                                                     random_state=10),
                                param_grid=param_test1, scoring='roc_auc', iid=False, cv=5)
        gsearch1.fit(x_train, y_train[flag_name])
        #gsearch1.grid_scores_, gsearch1.best_params_, gsearch1.best_score_

        #找到了一个合适的迭代次数，现在我们开始对决策树进行调参。首先我们对决策树最大深度max_depth和内部节点再划分所需最小样本数min_samples_split进行网格搜索
        param_test2 = {'max_depth': range(3, 14, 2), 'min_samples_split': range(100, 801, 200)}
        gsearch2 = GridSearchCV(
            estimator=GradientBoostingClassifier(learning_rate=0.1, n_estimators=60, min_samples_leaf=20,
                                                 max_features='sqrt', subsample=0.8, random_state=10),
            param_grid=param_test2, scoring='roc_auc', iid=False, cv=5)
        gsearch2.fit(x_train, y_train[flag_name])
        #gsearch2.grid_scores_, gsearch2.best_params_, gsearch2.best_score_

        #对于内部节点再划分所需最小样本数min_samples_split，我们暂时不能一起定下来，因为这个还和决策树其他的参数存在关联。
        # 下面我们再对内部节点再划分所需最小样本数min_samples_split和叶子节点最少样本数min_samples_leaf一起调参
        param_test3 = {'min_samples_split': range(800, 1900, 200), 'min_samples_leaf': range(60, 101, 10)}
        gsearch3 = GridSearchCV(estimator=GradientBoostingClassifier(learning_rate=0.1, n_estimators=60, max_depth=7,
                                                                     max_features='sqrt', subsample=0.8,
                                                                     random_state=10),
```



```
                                param_grid=param_test3, scoring='roc_auc', iid=False, cv=5)
        gsearch3.fit(x_train, y_train[flag_name])
        #gsearch3.grid_scores_, gsearch3.best_params_, gsearch3.best_score_

        #现在我们再对最大特征数max_features进行网格搜索
        param_test4 = {'max_features': range(7, 20, 2)}
        gsearch4 = GridSearchCV(
            estimator=GradientBoostingClassifier(learning_rate=0.1, n_estimators=60, max_depth=7, min_samples_leaf=60,
                                                 min_samples_split=1200, subsample=0.8, random_state=10),
            param_grid=param_test4, scoring='roc_auc', iid=False, cv=5)
        gsearch4.fit(x_train, y_train[flag_name])
        #gsearch4.grid_scores_, gsearch4.best_params_, gsearch4.best_score_

        #再对子采样的比例进行网格搜索
        param_test5 = {'subsample': [0.6, 0.7, 0.75, 0.8, 0.85, 0.9]}
        gsearch5 = GridSearchCV(
            estimator=GradientBoostingClassifier(learning_rate=0.1, n_estimators=60, max_depth=7, min_samples_leaf=60,
                                                 min_samples_split=1200, max_features=9, random_state=10),
            param_grid=param_test5, scoring='roc_auc', iid=False, cv=5)
        gsearch5.fit(x_train, y_train[flag_name])
        #gsearch5.grid_scores_, gsearch5.best_params_, gsearch5.best_score_

        #最后再对学习速率和最大迭代次数进行网格搜索
        param_test6 = {'learning_rate': [0.01, 0.02, 0.05, 0.1, 0.1, 0.2],'n_estimators':list(range(20,600,20))}
        gsearch6 = GridSearchCV(
            estimator=GradientBoostingClassifier(learning_rate=0.1, n_estimators=60, max_depth=7, min_samples_leaf=60,
                                                 min_samples_split=1200, max_features=9, random_state=10),
            param_grid=param_test5, scoring='roc_auc', iid=False, cv=5)
        gsearch6.fit(x_train, y_train[flag_name])
        #gsearch6.grid_scores_, gsearch6.best_params_, gsearch6.best_score_

    def standard_logistic_model(self):
        """
        Explanation
        ----------
        模型最终结果
        """
        t1=time.time()
        train_name = self.str_to_unicode(self.train_name)
        test_name = self.str_to_unicode(self.test_name)
        oot_name = self.str_to_unicode(self.oot_name)
        # feature analyse and woe
        if self.requirement=='online':
            train_data_bin, data_woe, oot_data_woe, train_data_stat, oot_data_stat, data_original, oot_data_original = self.trans_woe_multiprocess(
                self.data_total,
```

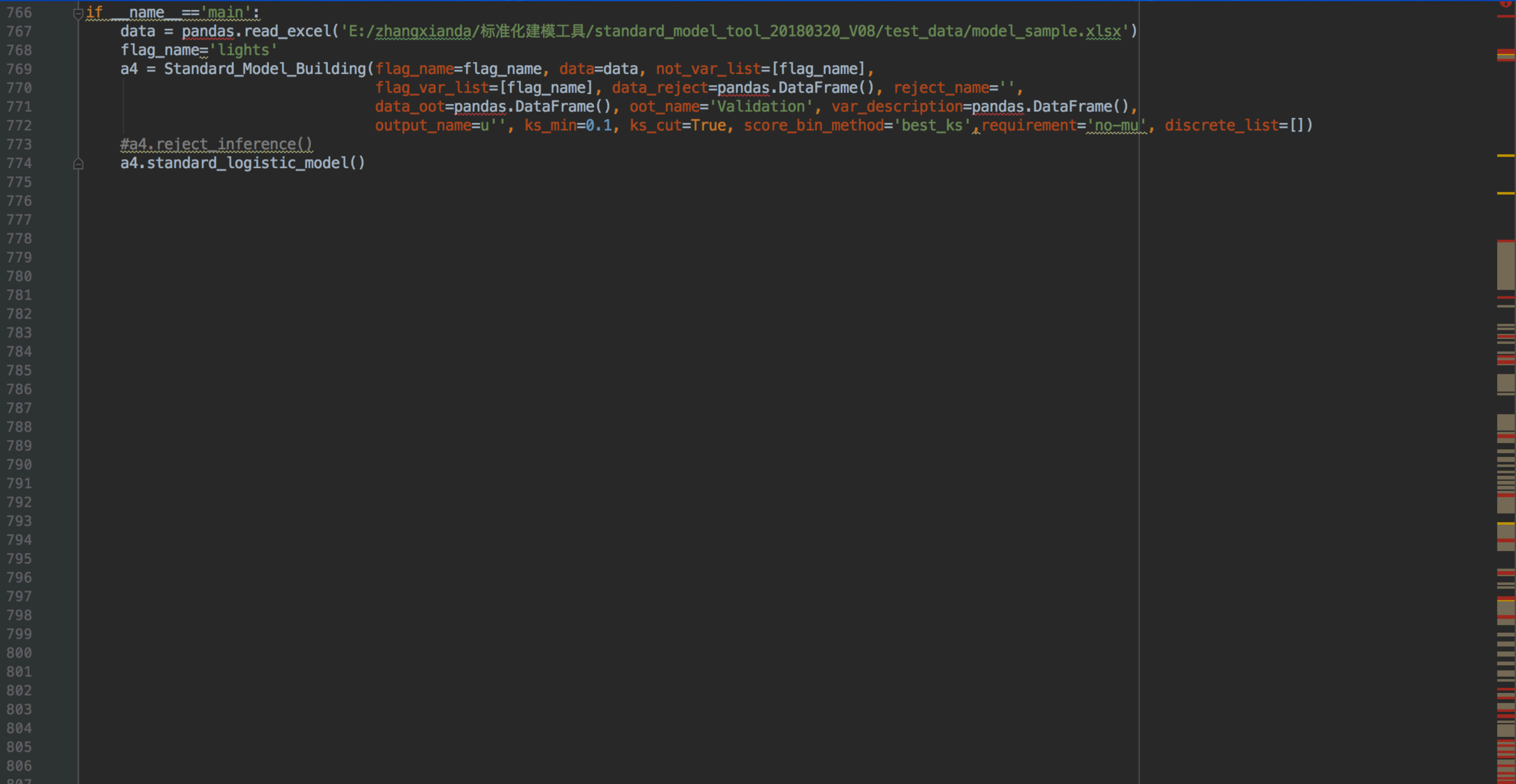


```
                pandas.DataFrame(),
                self.data_oot,
                self.train_name,
                self.test_name,
                self.oot_name)
        else:
            if self.requirement=='multiprocess':
                sk=self.trans_woe_multiprocess
            else:
                sk=self.trans_woe_no_multiprocess
            train_data_bin, data_woe, _, oot_data_woe, train_data_stat, _, oot_data_stat, data_original, _, oot_data_original = sk(
                self.data_total,
                pandas.DataFrame(),
                self.data_oot,
                self.train_name,
                self.test_name,
                self.oot_name)
        #feature select
        initial_feature_list,delete_dict=Base_Feature_Select_Rule.initial_feature_select(data_original,train_data_stat,self.discrete_list,self.single,self.iv,self
        furture_feature_list = Base_Feature_Select_Rule.further_feature_select(self.flag_name, self.not_var_list,
                                                                               train_data_stat[train_data_stat[
                                                                                   'Characteristic'].isin(
                                                                                   initial_feature_list)],
                                                                               data_woe[[self.flag_name] + initial_feature_list],
                                                                               self.C, self.selection_threshold,
                                                                               self.random_state, self.p_value)
        print(delete_dict)
        final_list=furture_feature_list+['intercept']
        # Model interation
        train_data_woe, test_data_woe, train_pre,test_pre,result = Standard_Model_Building.iteration_logistic(data_woe, self.flag_name,
                                                                                           self.train_rate, final_list,
                                                                                           self.ks_cut, self.ks_min,
                                                                                           self.ks_decline_rate)

        # oot and original data
        oot_data_woe['intercept']=1
        oot_pre = Standard_Model_Building.get_logit_pre(result, oot_data_woe, self.flag_name, final_list)
        train_data_original=data_original.loc[train_data_woe.index]
        test_data_original=data_original.loc[test_data_woe.index]
        # model evaluation
        flag_list = [self.flag_name]
        if not self.reject_name:
            print('Start without Reject Inference')
            tmp_pre = Summary_Of_Model_Result.model_score(train_pre, 'predict',train_name,'Logistic')
            if self.score_bin_method=='best_ks':
                score_bin = self.Best_Bin(self.flag_name, '%s_score' % train_name, tmp_pre, piece=10,rate=0.02,bin_thought=self.bin_thought)
```



```
            else:
                score_bin=Summary_Of_Model_Result.get_10_percentile(tmp_pre,self.flag_name,train_name,num=20)
                if len(score_bin)==0:
                    score_bin = self.Best_Bin(self.flag_name, '%s_score' % train_name, tmp_pre, piece=10)
            train_data_stat.insert(1,'Sample','Development Sample')
            if len(oot_data_stat)>0:
                oot_data_stat.insert(1, 'Sample', oot_name)
            # combine result
            Summary_Of_Model_Result.combine_model_result(self.flag_name,self.not_var_list,[],self.reject_name,self.var_description,score_bin , train_data_bin, trai
                                 train_data_stat,
                                 oot_data_stat, train_data_original, test_data_original, oot_data_original,
                                 self.output_path,self.output_name,train_name,test_name, oot_name,self.not_in_list,'Logistic',result,train_data_woe,test_data_woe,o
            print(time.time() - t1)
        else:
            print('Start with Reject Inference')
            flag_list.append(self.reject_name)
            agb_data_woe=data_woe
            agb_data_original=data_original
            kgb_data_woe=agb_data_woe[agb_data_woe[self.reject_name]==0]
            kgb_data_original=agb_data_original[agb_data_original[self.reject_name]==0]
            #pre
            agb_pre = pandas.concat([train_pre, test_pre])
            kgb_pre=Standard_Model_Building.get_logit_pre(result,kgb_data_woe,self.flag_name,final_list)
            #stat
            agb_stat=train_data_stat
            kgb_stat = [self.get_data_stat_else(kgb_data_original, kgb_data_woe, self.flag_name, var,
                                           train_data_bin[train_data_bin['Characteristic'] == var][
                                               'Char_Type'].max()) for var in furture_feature_list]
            kgb_stat = pandas.DataFrame(kgb_stat)
            agb_stat.insert(1,'Sample','Development - AGB')
            kgb_stat.insert(1, 'Sample', 'Development - KGB')
            if len(oot_data_stat)>0:
                oot_data_stat.insert(1, 'Sample', oot_name)
            tmp_pre = Summary_Of_Model_Result.model_score(agb_pre, 'predict','Development - AGB','Logistic')
            if self.score_bin_method=='best_ks':
                score_bin = self.Best_Bin(self.flag_name, 'Development - AGB_score' , tmp_pre, piece=10)
            else:
                score_bin=Summary_Of_Model_Result.get_10_percentile(tmp_pre,self.flag_name,'Development - AGB_score',num=20)
            # combine result
            Summary_Of_Model_Result.combine_model_result(self.flag_name,self.not_var_list,[],self.reject_name, self.var_description,score_bin,train_data_bin, agb
                                      agb_stat,
                                      kgb_stat,
                                      oot_data_stat, agb_data_original, kgb_data_original, oot_data_original,
                                      self.output_path,self.output_name, 'Development - AGB', 'Development - KGB', oot_name, self.not_in_list,
                                    'Logistic',result,agb_data_woe,kgb_data_woe,oot_data_woe)
```

调用示例



```python
if __name__=='main':
    data = pandas.read_excel('E:/zhangxianda/标准化建模工具/standard_model_tool_20180320_V08/test_data/model_sample.xlsx')
    flag_name='lights'
    a4 = Standard_Model_Building(flag_name=flag_name, data=data, not_var_list=[flag_name],
                                 flag_var_list=[flag_name], data_reject=pandas.DataFrame(), reject_name='',
                                 data_oot=pandas.DataFrame(), oot_name='Validation', var_description=pandas.DataFrame(),
                                 output_name=u'', ks_min=0.1, ks_cut=True, score_bin_method='best_ks',requirement='no-mu', discrete_list=[])
    #a4.reject_inference()
    a4.standard_logistic_model()
```

5

code demo

模型评估

模型评估结果

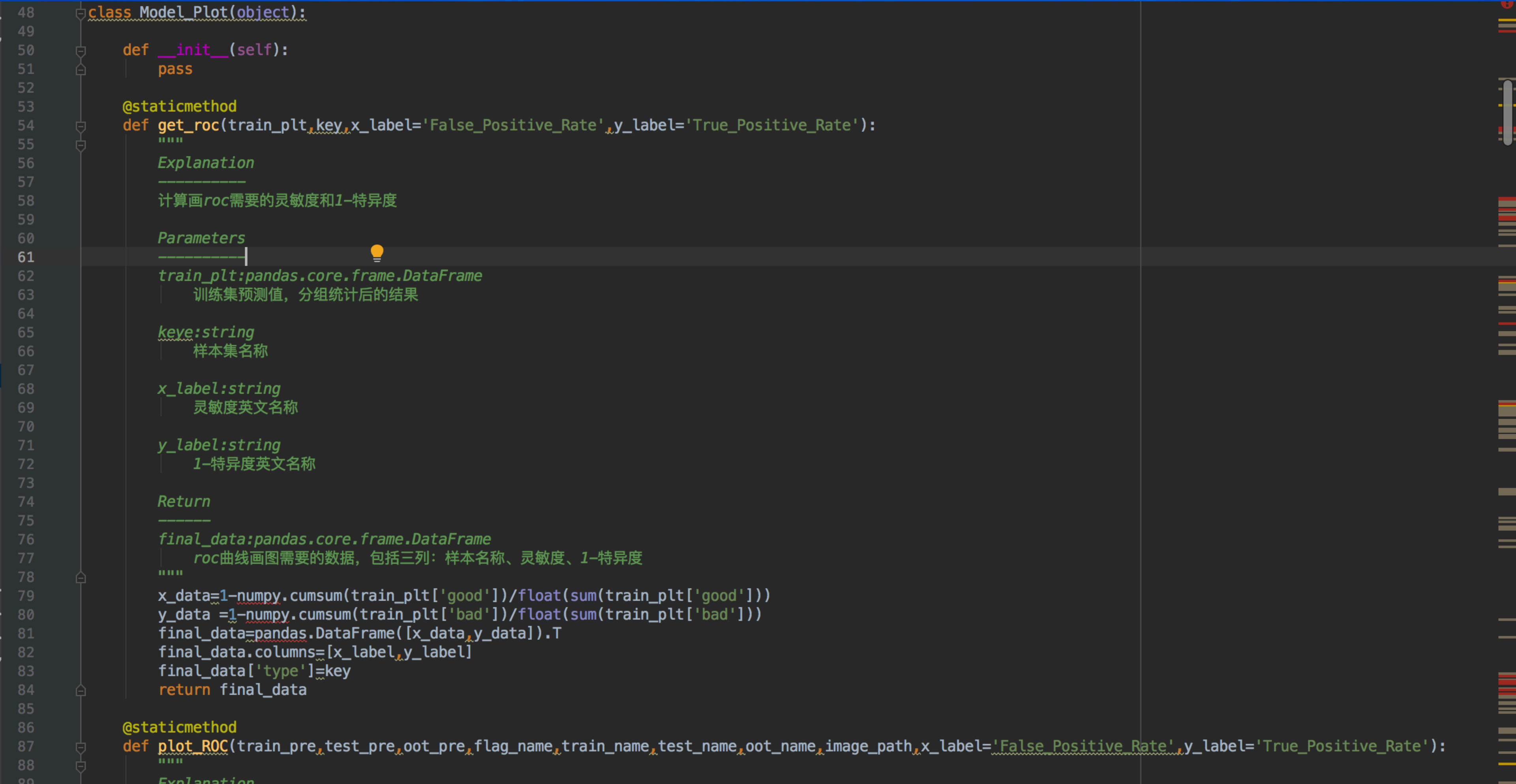
图文类

1. **ROC**图
2. **KS**图
3. **PR-F1**图
4. 模型分数分布图
5. ……

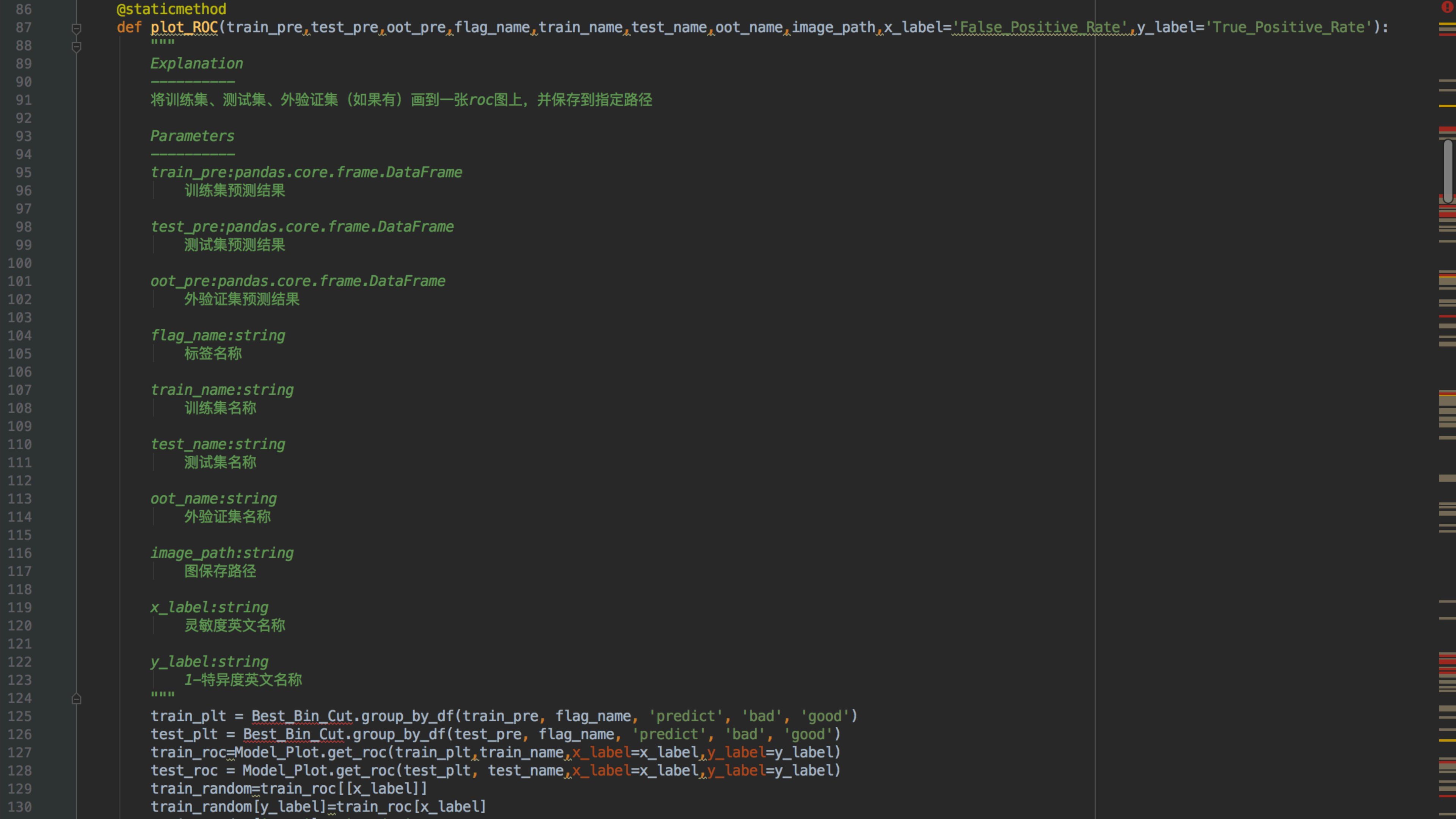
表格类

1. 训练集、测试集、外验证集的**KS、ROC、GINI、AR、F1、Precision、Recall**等统计
2. 模型分数分布表
3. **PSI**统计表(模型分数&变量)
4. Swap set
5. ……

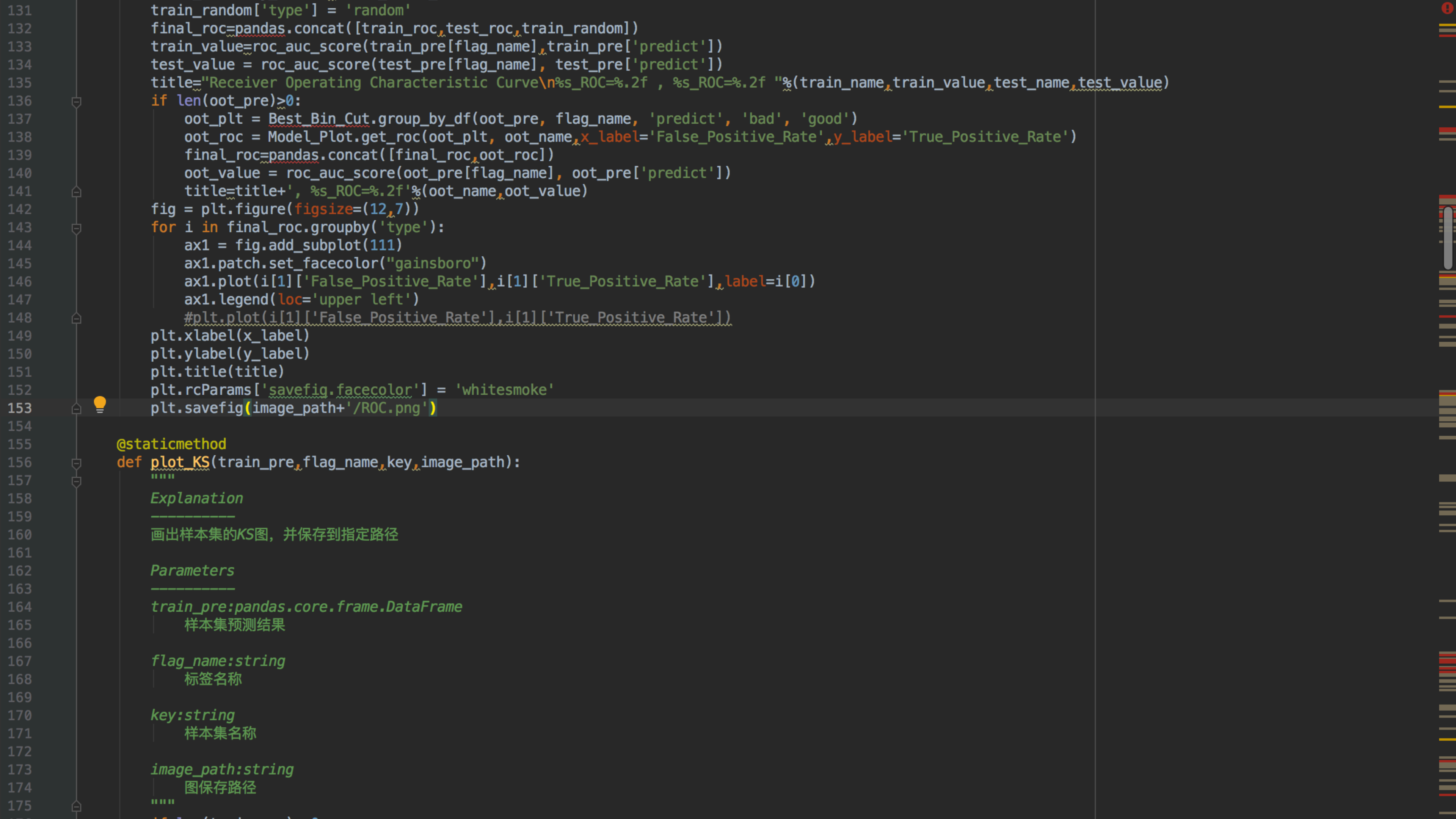
1 模型评估-画图



```python
class Model_Plot(object):

    def __init__(self):
        pass

    @staticmethod
    def get_roc(train_plt,key,x_label='False_Positive_Rate',y_label='True_Positive_Rate'):
        """
        Explanation
        ----------
        计算画roc需要的灵敏度和1-特异度

        Parameters
        ----------
        train_plt:pandas.core.frame.DataFrame
            训练集预测值，分组统计后的结果

        keye:string
            样本集名称

        x_label:string
            灵敏度英文名称

        y_label:string
            1-特异度英文名称

        Return
        -------
        final_data:pandas.core.frame.DataFrame
            roc曲线画图需要的数据，包括三列：样本名称、灵敏度、1-特异度
        """
        x_data=1-numpy.cumsum(train_plt['good'])/float(sum(train_plt['good']))
        y_data =1-numpy.cumsum(train_plt['bad'])/float(sum(train_plt['bad']))
        final_data=pandas.DataFrame([x_data,y_data]).T
        final_data.columns=[x_label,y_label]
        final_data['type']=key
        return final_data

    @staticmethod
    def plot_ROC(train_pre,test_pre,oot_pre,flag_name,train_name,test_name,oot_name,image_path,x_label='False_Positive_Rate',y_label='True_Positive_Rate'):
        """
        Explanation
```

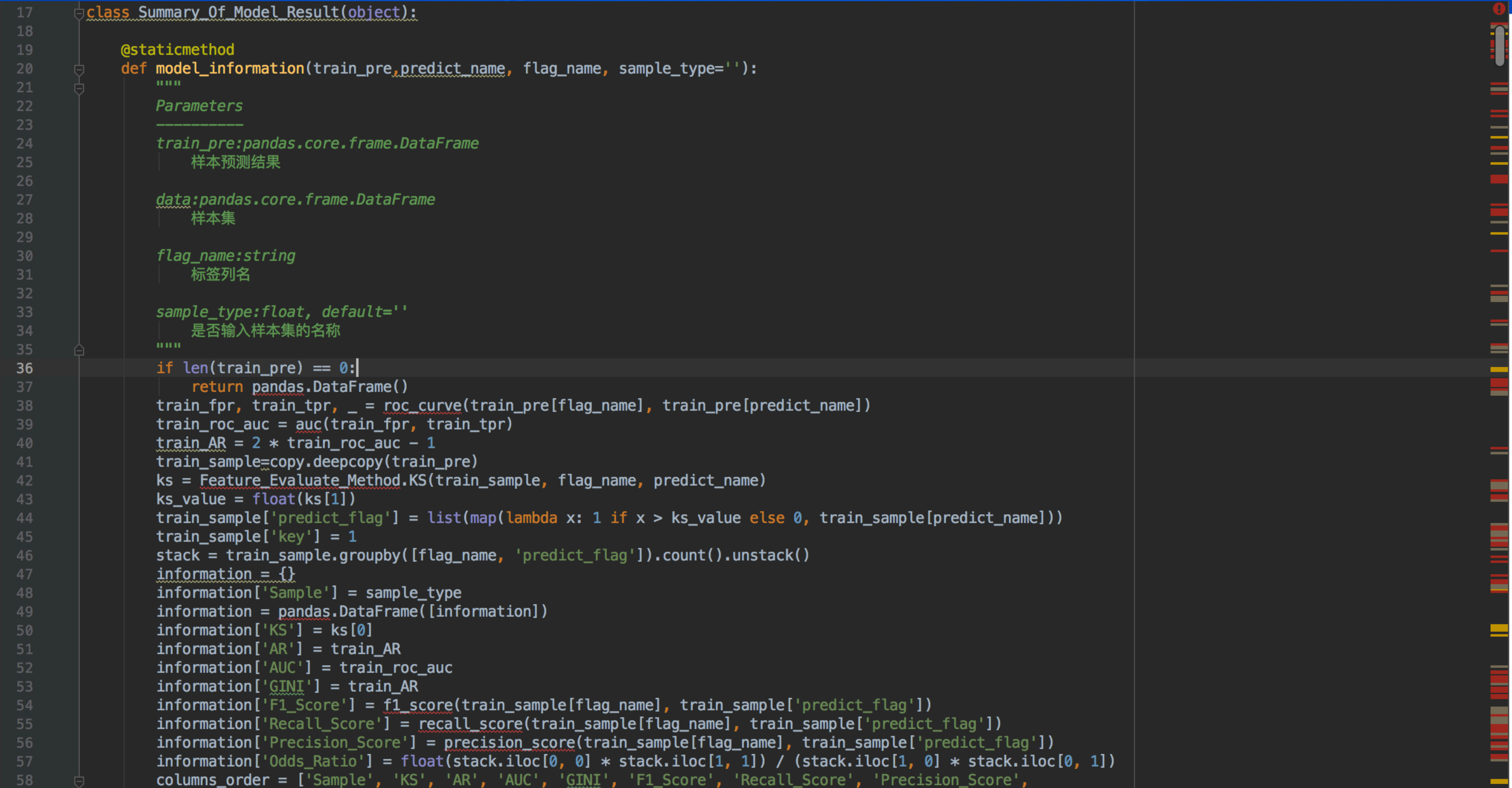


```python
    @staticmethod
    def plot_ROC(train_pre,test_pre,oot_pre,flag_name,train_name,test_name,oot_name,image_path,x_label='False_Positive_Rate',y_label='True_Positive_Rate'):
        """
        Explanation
        -----------
        将训练集、测试集、外验证集（如果有）画到一张roc图上，并保存到指定路径

        Parameters
        -----------
        train_pre:pandas.core.frame.DataFrame
            训练集预测结果

        test_pre:pandas.core.frame.DataFrame
            测试集预测结果

        oot_pre:pandas.core.frame.DataFrame
            外验证集预测结果

        flag_name:string
            标签名称

        train_name:string
            训练集名称

        test_name:string
            测试集名称

        oot_name:string
            外验证集名称

        image_path:string
            图保存路径

        x_label:string
            灵敏度英文名称

        y_label:string
            1-特异度英文名称
        """
        train_plt = Best_Bin_Cut.group_by_df(train_pre, flag_name, 'predict', 'bad', 'good')
        test_plt = Best_Bin_Cut.group_by_df(test_pre, flag_name, 'predict', 'bad', 'good')
        train_roc=Model_Plot.get_roc(train_plt,train_name,x_label=x_label,y_label=y_label)
        test_roc = Model_Plot.get_roc(test_plt, test_name,x_label=x_label,y_label=y_label)
        train_random=train_roc[[x_label]]
        train_random[y_label]=train_roc[x_label]
```

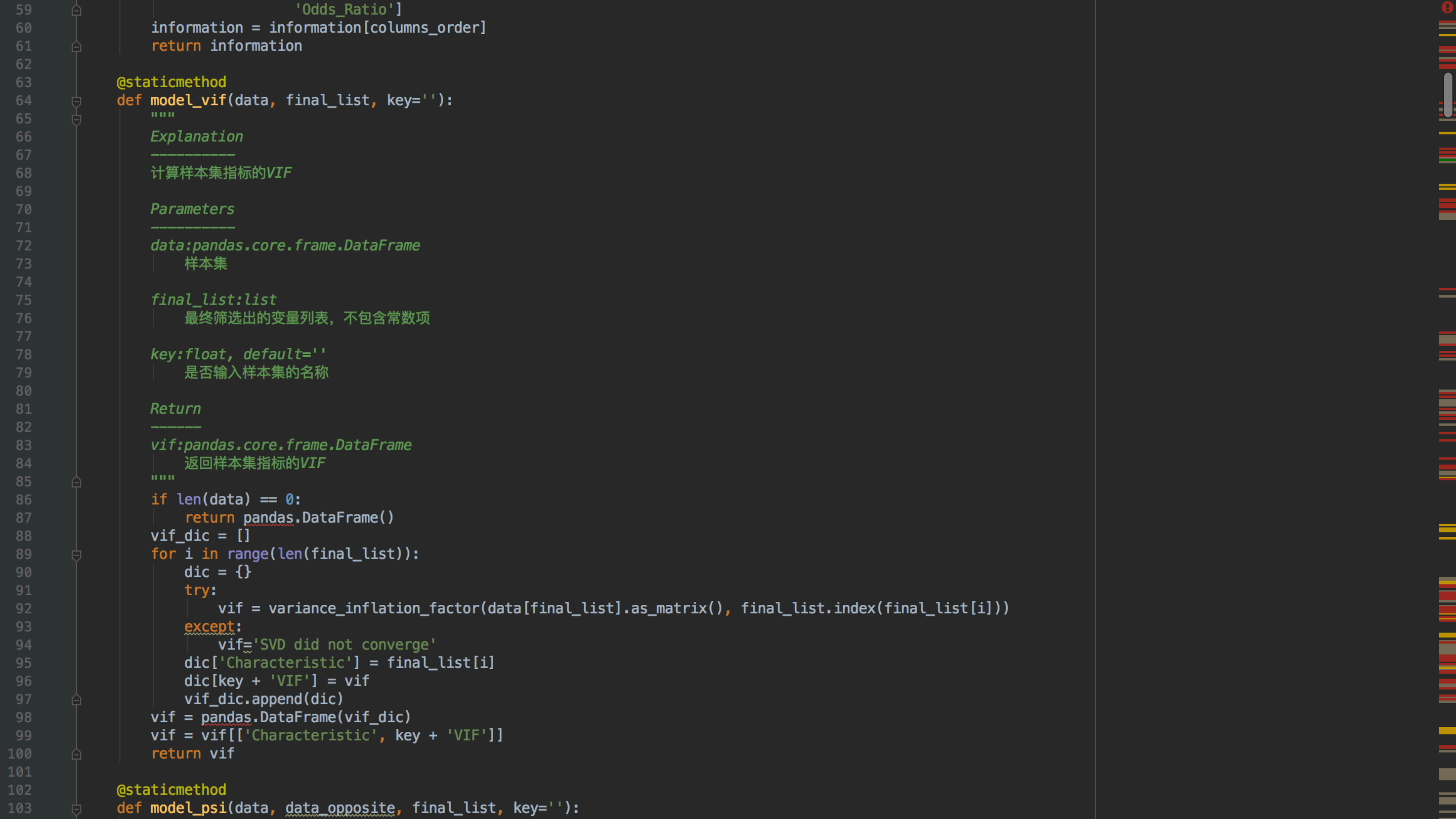


```
        train_random['type'] = 'random'
        final_roc=pandas.concat([train_roc,test_roc,train_random])
        train_value=roc_auc_score(train_pre[flag_name],train_pre['predict'])
        test_value = roc_auc_score(test_pre[flag_name], test_pre['predict'])
        title="Receiver Operating Characteristic Curve\n%s_ROC=%.2f , %s_ROC=%.2f "%(train_name,train_value,test_name,test_value)
        if len(oot_pre)>0:
            oot_plt = Best_Bin_Cut.group_by_df(oot_pre, flag_name, 'predict', 'bad', 'good')
            oot_roc = Model_Plot.get_roc(oot_plt, oot_name,x_label='False_Positive_Rate',y_label='True_Positive_Rate')
            final_roc=pandas.concat([final_roc,oot_roc])
            oot_value = roc_auc_score(oot_pre[flag_name], oot_pre['predict'])
            title=title+', %s_ROC=%.2f'%(oot_name,oot_value)
        fig = plt.figure(figsize=(12,7))
        for i in final_roc.groupby('type'):
            ax1 = fig.add_subplot(111)
            ax1.patch.set_facecolor("gainsboro")
            ax1.plot(i[1]['False_Positive_Rate'],i[1]['True_Positive_Rate'],label=i[0])
            ax1.legend(loc='upper left')
            #plt.plot(i[1]['False_Positive_Rate'],i[1]['True_Positive_Rate'])
        plt.xlabel(x_label)
        plt.ylabel(y_label)
        plt.title(title)
        plt.rcParams['savefig.facecolor'] = 'whitesmoke'
        plt.savefig(image_path+'/ROC.png')

    @staticmethod
    def plot_KS(train_pre,flag_name,key,image_path):
        """
        Explanation
        ----------
        画出样本集的KS图，并保存到指定路径

        Parameters
        ----------
        train_pre:pandas.core.frame.DataFrame
            样本集预测结果

        flag_name:string
            标签名称

        key:string
            样本集名称

        image_path:string
            图保存路径
        """
```

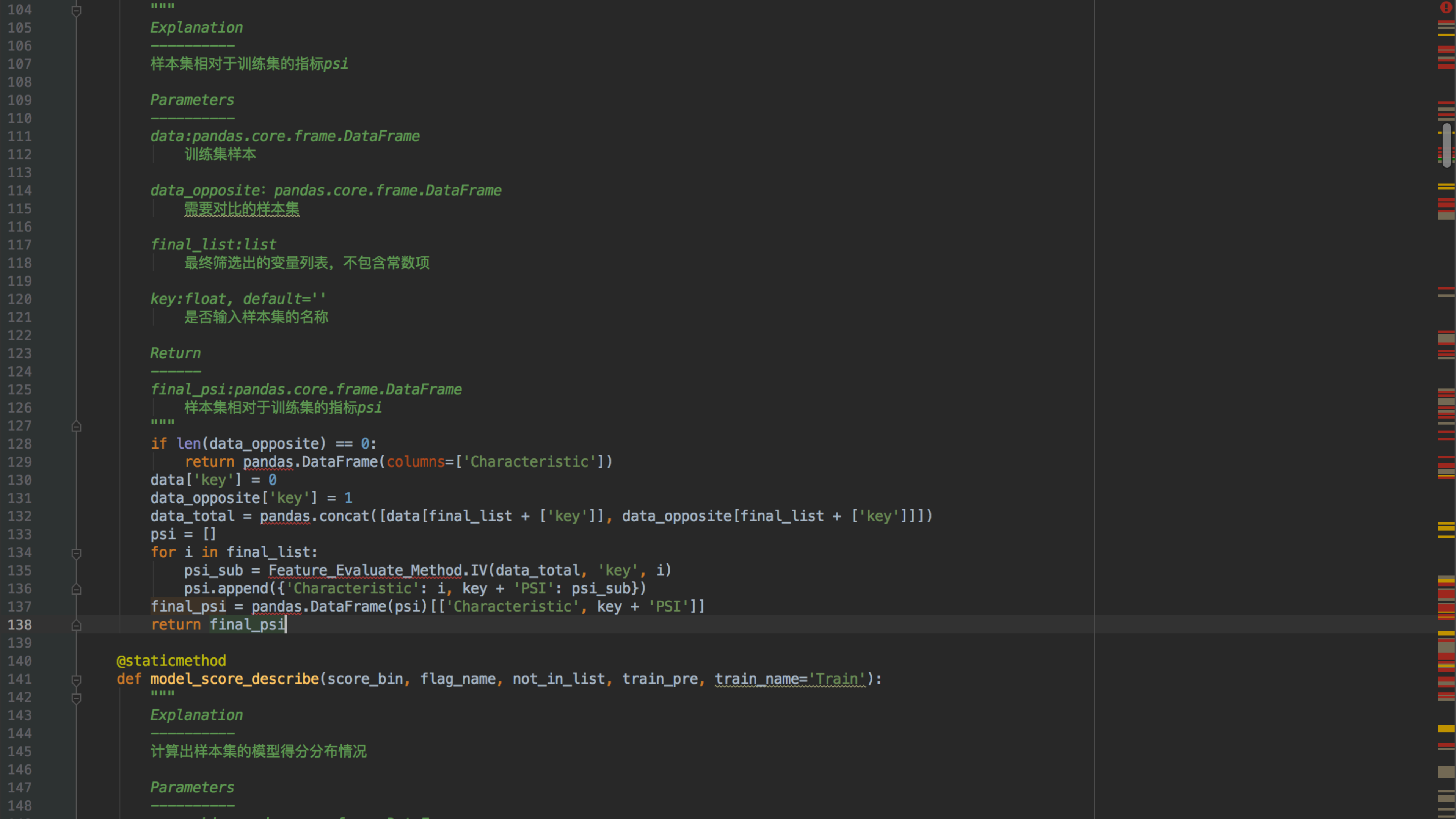
②模型评估-表格统计



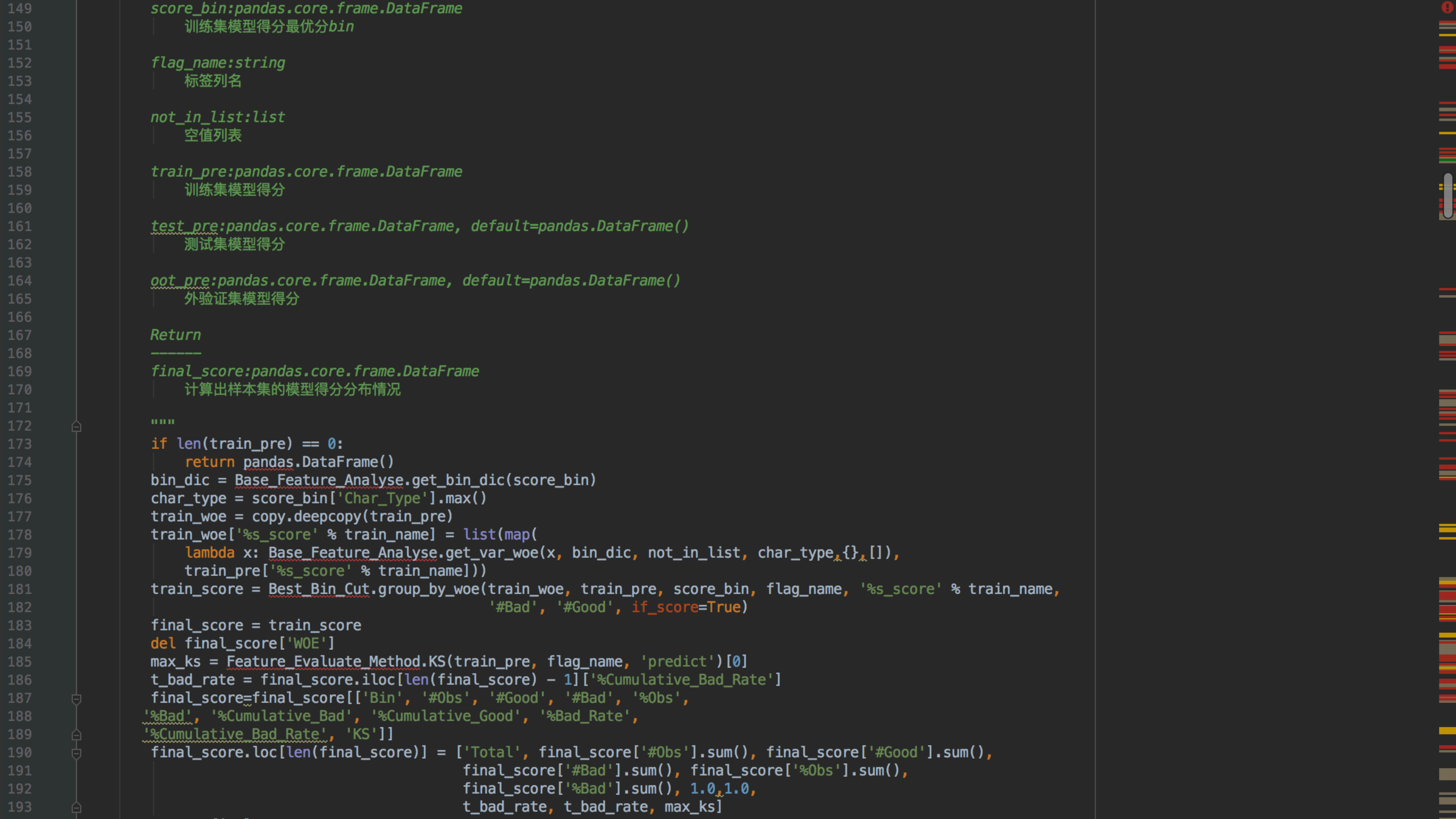
```
class Summary_Of_Model_Result(object):

    @staticmethod
    def model_information(train_pre,predict_name, flag_name, sample_type=''):
        """
        Parameters
        ------------
        train_pre:pandas.core.frame.DataFrame
            样本预测结果

        data:pandas.core.frame.DataFrame
            样本集

        flag_name:string
            标签列名

        sample_type:float, default=''
            是否输入样本集的名称
        """
        if len(train_pre) == 0:
            return pandas.DataFrame()
        train_fpr, train_tpr, _ = roc_curve(train_pre[flag_name], train_pre[predict_name])
        train_roc_auc = auc(train_fpr, train_tpr)
        train_AR = 2 * train_roc_auc - 1
        train_sample=copy.deepcopy(train_pre)
        ks = Feature_Evaluate_Method.KS(train_sample, flag_name, predict_name)
        ks_value = float(ks[1])
        train_sample['predict_flag'] = list(map(lambda x: 1 if x > ks_value else 0, train_sample[predict_name]))
        train_sample['key'] = 1
        stack = train_sample.groupby([flag_name, 'predict_flag']).count().unstack()
        information = {}
        information['Sample'] = sample_type
        information = pandas.DataFrame([information])
        information['KS'] = ks[0]
        information['AR'] = train_AR
        information['AUC'] = train_roc_auc
        information['GINI'] = train_AR
        information['F1_Score'] = f1_score(train_sample[flag_name], train_sample['predict_flag'])
        information['Recall_Score'] = recall_score(train_sample[flag_name], train_sample['predict_flag'])
        information['Precision_Score'] = precision_score(train_sample[flag_name], train_sample['predict_flag'])
        information['Odds_Ratio'] = float(stack.iloc[0, 0] * stack.iloc[1, 1]) / (stack.iloc[1, 0] * stack.iloc[0, 1])
        columns_order = ['Sample', 'KS', 'AR', 'AUC', 'GINI', 'F1_Score', 'Recall_Score', 'Precision_Score',
```



```
                         'Odds_Ratio']
        information = information[columns_order]
        return information

    @staticmethod
    def model_vif(data, final_list, key=''):
        """
        Explanation
        ----------
        计算样本集指标的VIF

        Parameters
        ----------
        data:pandas.core.frame.DataFrame
            样本集

        final_list:list
            最终筛选出的变量列表，不包含常数项

        key:float, default=''
            是否输入样本集的名称

        Return
        ------
        vif:pandas.core.frame.DataFrame
            返回样本集指标的VIF
        """
        if len(data) == 0:
            return pandas.DataFrame()
        vif_dic = []
        for i in range(len(final_list)):
            dic = {}
            try:
                vif = variance_inflation_factor(data[final_list].as_matrix(), final_list.index(final_list[i]))
            except:
                vif='SVD did not converge'
            dic['Characteristic'] = final_list[i]
            dic[key + 'VIF'] = vif
            vif_dic.append(dic)
        vif = pandas.DataFrame(vif_dic)
        vif = vif[['Characteristic', key + 'VIF']]
        return vif

    @staticmethod
    def model_psi(data, data_opposite, final_list, key=''):
```



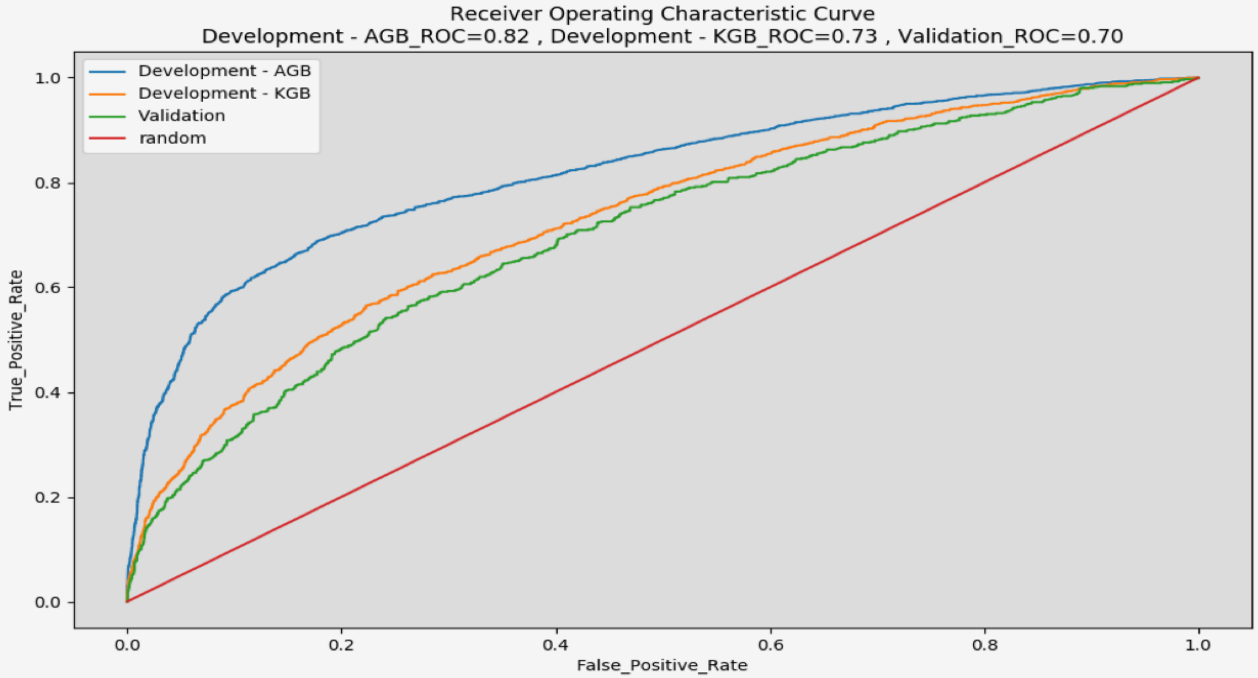
```
        """
        Explanation
        -----------
        样本集相对于训练集的指标psi

        Parameters
        -----------
        data:pandas.core.frame.DataFrame
            训练集样本

        data_opposite： pandas.core.frame.DataFrame
            需要对比的样本集

        final_list:list
            最终筛选出的变量列表，不包含常数项

        key:float, default=''
            是否输入样本集的名称

        Return
        -------
        final_psi:pandas.core.frame.DataFrame
            样本集相对于训练集的指标psi
        """
        if len(data_opposite) == 0:
            return pandas.DataFrame(columns=['Characteristic'])
        data['key'] = 0
        data_opposite['key'] = 1
        data_total = pandas.concat([data[final_list + ['key']], data_opposite[final_list + ['key']]])
        psi = []
        for i in final_list:
            psi_sub = Feature_Evaluate_Method.IV(data_total, 'key', i)
            psi.append({'Characteristic': i, key + 'PSI': psi_sub})
        final_psi = pandas.DataFrame(psi)[['Characteristic', key + 'PSI']]
        return final_psi

    @staticmethod
    def model_score_describe(score_bin, flag_name, not_in_list, train_pre, train_name='Train'):
        """
        Explanation
        -----------
        计算出样本集的模型得分分布情况

        Parameters
        -----------
```



```
    score_bin:pandas.core.frame.DataFrame
        训练集模型得分最优分bin

    flag_name:string
        标签列名

    not_in_list:list
        空值列表

    train_pre:pandas.core.frame.DataFrame
        训练集模型得分

    test_pre:pandas.core.frame.DataFrame, default=pandas.DataFrame()
        测试集模型得分

    oot_pre:pandas.core.frame.DataFrame, default=pandas.DataFrame()
        外验证集模型得分

    Return
    ------
    final_score:pandas.core.frame.DataFrame
        计算出样本集的模型得分分布情况

    """
    if len(train_pre) == 0:
        return pandas.DataFrame()
    bin_dic = Base_Feature_Analyse.get_bin_dic(score_bin)
    char_type = score_bin['Char_Type'].max()
    train_woe = copy.deepcopy(train_pre)
    train_woe['%s_score' % train_name] = list(map(
        lambda x: Base_Feature_Analyse.get_var_woe(x, bin_dic, not_in_list, char_type,{},[]),
        train_pre['%s_score' % train_name]))
    train_score = Best_Bin_Cut.group_by_woe(train_woe, train_pre, score_bin, flag_name, '%s_score' % train_name,
                                            '#Bad', '#Good', if_score=True)
    final_score = train_score
    del final_score['WOE']
    max_ks = Feature_Evaluate_Method.KS(train_pre, flag_name, 'predict')[0]
    t_bad_rate = final_score.iloc[len(final_score) - 1]['%Cumulative_Bad_Rate']
    final_score=final_score[['Bin', '#Obs', '#Good', '#Bad', '%Obs',
   '%Bad', '%Cumulative_Bad', '%Cumulative_Good', '%Bad_Rate',
   '%Cumulative_Bad_Rate', 'KS']]
    final_score.loc[len(final_score)] = ['Total', final_score['#Obs'].sum(), final_score['#Good'].sum(),
                                         final_score['#Bad'].sum(), final_score['%Obs'].sum(),
                                         final_score['%Bad'].sum(), 1.0,1.0,
                                         t_bad_rate, t_bad_rate, max_ks]
```

结果展示-画图

ROC Curve

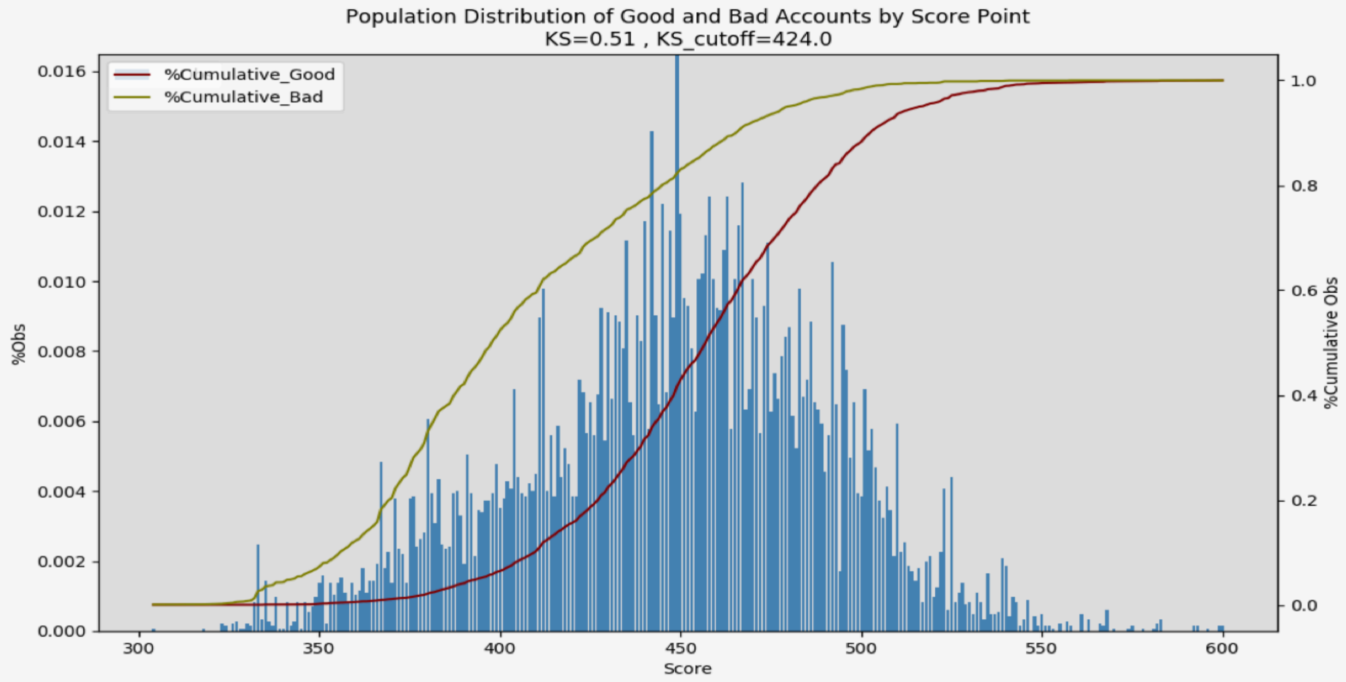


结果展示-画图

同盾大学
xue.tongdun.cn

Ascending Score Report

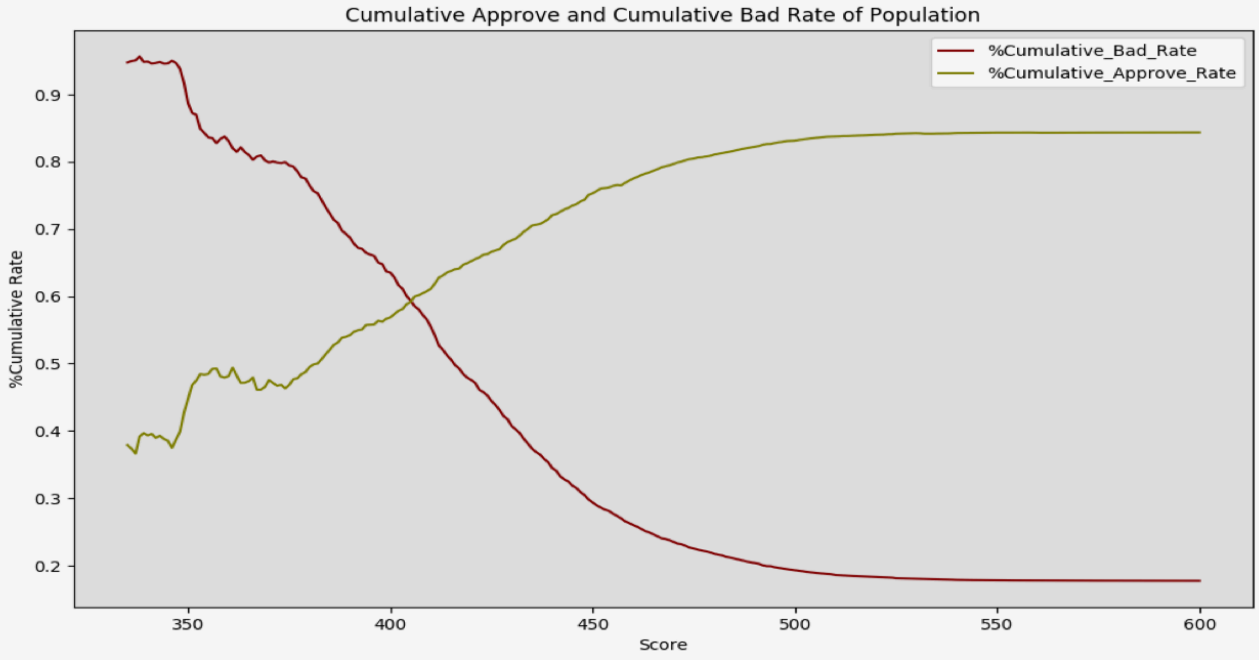
a. Population Distribution of Good and Bad Accounts by Score Point



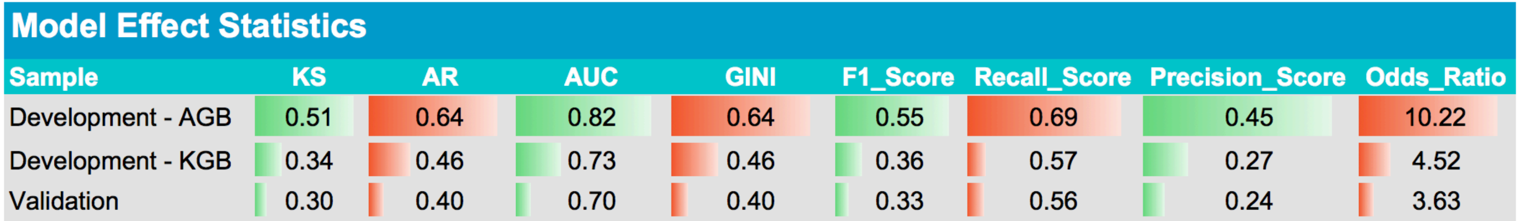
结果展示-画图

Ascending Score Report

b. Cumulative Approval and Cumulative Bad Rate of Population

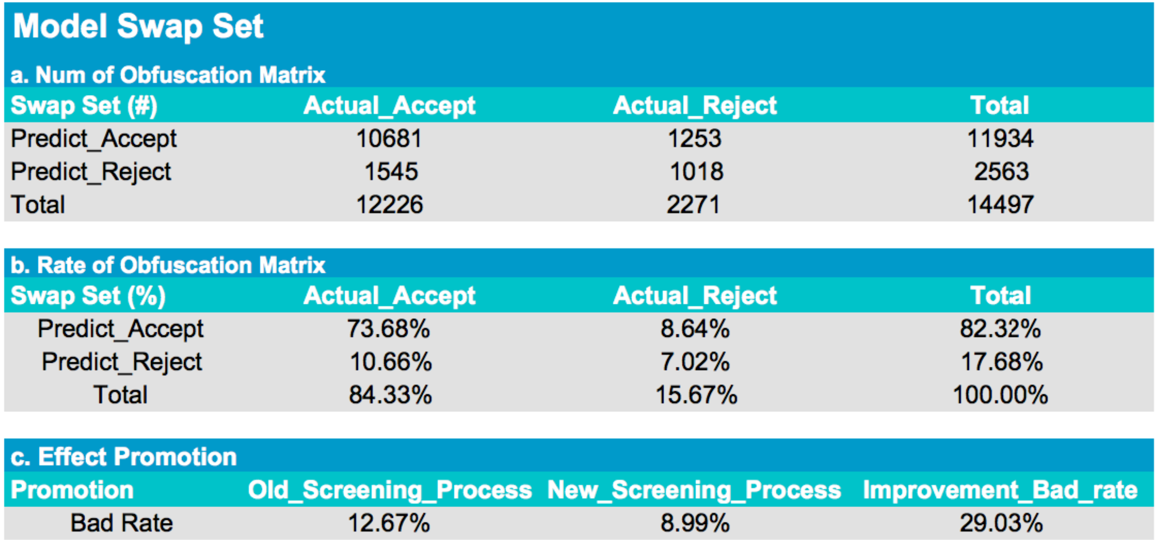


结果展示-表格统计



Model Effect Statistics

Sample	KS	AR	AUC	GINI	F1_Score	Recall_Score	Precision_Score	Odds_Ratio
Development - AGB	0.51	0.64	0.82	0.64	0.55	0.69	0.45	10.22
Development - KGB	0.34	0.46	0.73	0.46	0.36	0.57	0.27	4.52
Validation	0.30	0.40	0.70	0.40	0.33	0.56	0.24	3.63



Model Swap Set

a. Num of Obfuscation Matrix

Swap Set (#)	Actual_Accept	Actual_Reject	Total
Predict_Accept	10681	1253	11934
Predict_Reject	1545	1018	2563
Total	12226	2271	14497

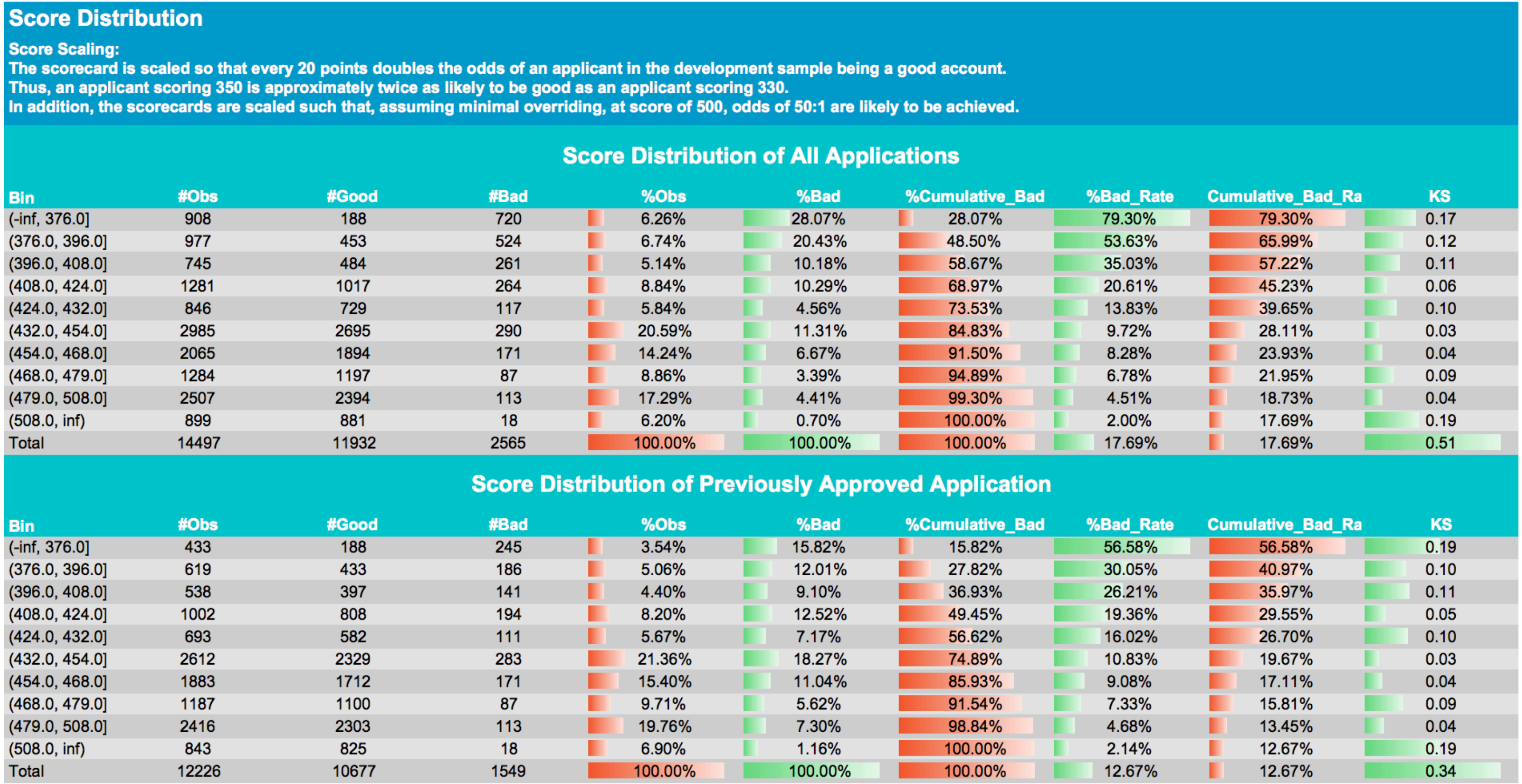
b. Rate of Obfuscation Matrix

Swap Set (%)	Actual_Accept	Actual_Reject	Total
Predict_Accept	73.68%	8.64%	82.32%
Predict_Reject	10.66%	7.02%	17.68%
Total	84.33%	15.67%	100.00%

c. Effect Promotion

Promotion	Old_Screening_Process	New_Screening_Process	Improvement_Bad_rate
Bad Rate	12.67%	8.99%	29.03%

结果展示-表格统计



Score Distribution

Score Scaling:
The scorecard is scaled so that every 20 points doubles the odds of an applicant in the development sample being a good account.
Thus, an applicant scoring 350 is approximately twice as likely to be good as an applicant scoring 330.
In addition, the scorecards are scaled such that, assuming minimal overriding, at score of 500, odds of 50:1 are likely to be achieved.

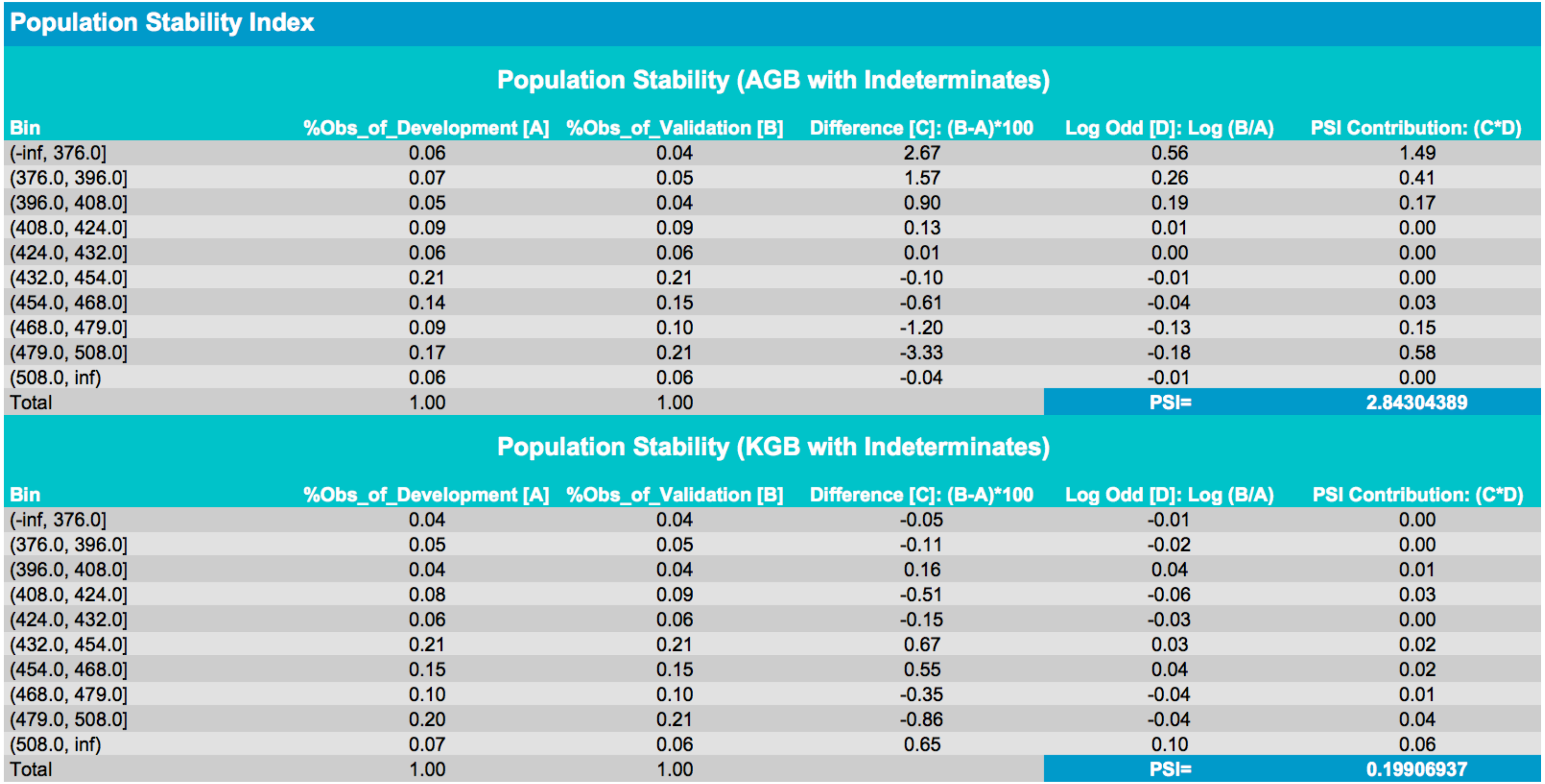
Score Distribution of All Applications

Bin	#Obs	#Good	#Bad	%Obs	%Bad	%Cumulative_Bad	%Bad_Rate	Cumulative_Bad_Ra	KS
(-inf, 376.0]	908	188	720	6.26%	28.07%	28.07%	79.30%	79.30%	0.17
(376.0, 396.0]	977	453	524	6.74%	20.43%	48.50%	53.63%	65.99%	0.12
(396.0, 408.0]	745	484	261	5.14%	10.18%	58.67%	35.03%	57.22%	0.11
(408.0, 424.0]	1281	1017	264	8.84%	10.29%	68.97%	20.61%	45.23%	0.06
(424.0, 432.0]	846	729	117	5.84%	4.56%	73.53%	13.83%	39.65%	0.10
(432.0, 454.0]	2985	2695	290	20.59%	11.31%	84.83%	9.72%	28.11%	0.03
(454.0, 468.0]	2065	1894	171	14.24%	6.67%	91.50%	8.28%	23.93%	0.04
(468.0, 479.0]	1284	1197	87	8.86%	3.39%	94.89%	6.78%	21.95%	0.09
(479.0, 508.0]	2507	2394	113	17.29%	4.41%	99.30%	4.51%	18.73%	0.04
(508.0, inf)	899	881	18	6.20%	0.70%	100.00%	2.00%	17.69%	0.19
Total	14497	11932	2565	100.00%	100.00%	100.00%	17.69%	17.69%	0.51

Score Distribution of Previously Approved Application

Bin	#Obs	#Good	#Bad	%Obs	%Bad	%Cumulative_Bad	%Bad_Rate	Cumulative_Bad_Ra	KS
(-inf, 376.0]	433	188	245	3.54%	15.82%	15.82%	56.58%	56.58%	0.19
(376.0, 396.0]	619	433	186	5.06%	12.01%	27.82%	30.05%	40.97%	0.10
(396.0, 408.0]	538	397	141	4.40%	9.10%	36.93%	26.21%	35.97%	0.11
(408.0, 424.0]	1002	808	194	8.20%	12.52%	49.45%	19.36%	29.55%	0.05
(424.0, 432.0]	693	582	111	5.67%	7.17%	56.62%	16.02%	26.70%	0.10
(432.0, 454.0]	2612	2329	283	21.36%	18.27%	74.89%	10.83%	19.67%	0.03
(454.0, 468.0]	1883	1712	171	15.40%	11.04%	85.93%	9.08%	17.11%	0.04
(468.0, 479.0]	1187	1100	87	9.71%	5.62%	91.54%	7.33%	15.81%	0.09
(479.0, 508.0]	2416	2303	113	19.76%	7.30%	98.84%	4.68%	13.45%	0.04
(508.0, inf)	843	825	18	6.90%	1.16%	100.00%	2.14%	12.67%	0.19
Total	12226	10677	1549	100.00%	100.00%	100.00%	12.67%	12.67%	0.34

结果展示-表格统计



Population Stability Index

Population Stability (AGB with Indeterminates)

Bin	%Obs_of_Development [A]	%Obs_of_Validation [B]	Difference [C]: (B-A)*100	Log Odd [D]: Log (B/A)	PSI Contribution: (C*D)
(-inf, 376.0]	0.06	0.04	2.67	0.56	1.49
(376.0, 396.0]	0.07	0.05	1.57	0.26	0.41
(396.0, 408.0]	0.05	0.04	0.90	0.19	0.17
(408.0, 424.0]	0.09	0.09	0.13	0.01	0.00
(424.0, 432.0]	0.06	0.06	0.01	0.00	0.00
(432.0, 454.0]	0.21	0.21	-0.10	-0.01	0.00
(454.0, 468.0]	0.14	0.15	-0.61	-0.04	0.03
(468.0, 479.0]	0.09	0.10	-1.20	-0.13	0.15
(479.0, 508.0]	0.17	0.21	-3.33	-0.18	0.58
(508.0, inf)	0.06	0.06	-0.04	-0.01	0.00
Total	1.00	1.00		PSI=	2.84304389

Population Stability (KGB with Indeterminates)

Bin	%Obs_of_Development [A]	%Obs_of_Validation [B]	Difference [C]: (B-A)*100	Log Odd [D]: Log (B/A)	PSI Contribution: (C*D)
(-inf, 376.0]	0.04	0.04	-0.05	-0.01	0.00
(376.0, 396.0]	0.05	0.05	-0.11	-0.02	0.00
(396.0, 408.0]	0.04	0.04	0.16	0.04	0.01
(408.0, 424.0]	0.08	0.09	-0.51	-0.06	0.03
(424.0, 432.0]	0.06	0.06	-0.15	-0.03	0.00
(432.0, 454.0]	0.21	0.21	0.67	0.03	0.02
(454.0, 468.0]	0.15	0.15	0.55	0.04	0.02
(468.0, 479.0]	0.10	0.10	-0.35	-0.04	0.01
(479.0, 508.0]	0.20	0.21	-0.86	-0.04	0.04
(508.0, inf)	0.07	0.06	0.65	0.10	0.06
Total	1.00	1.00		PSI=	0.19906937

6

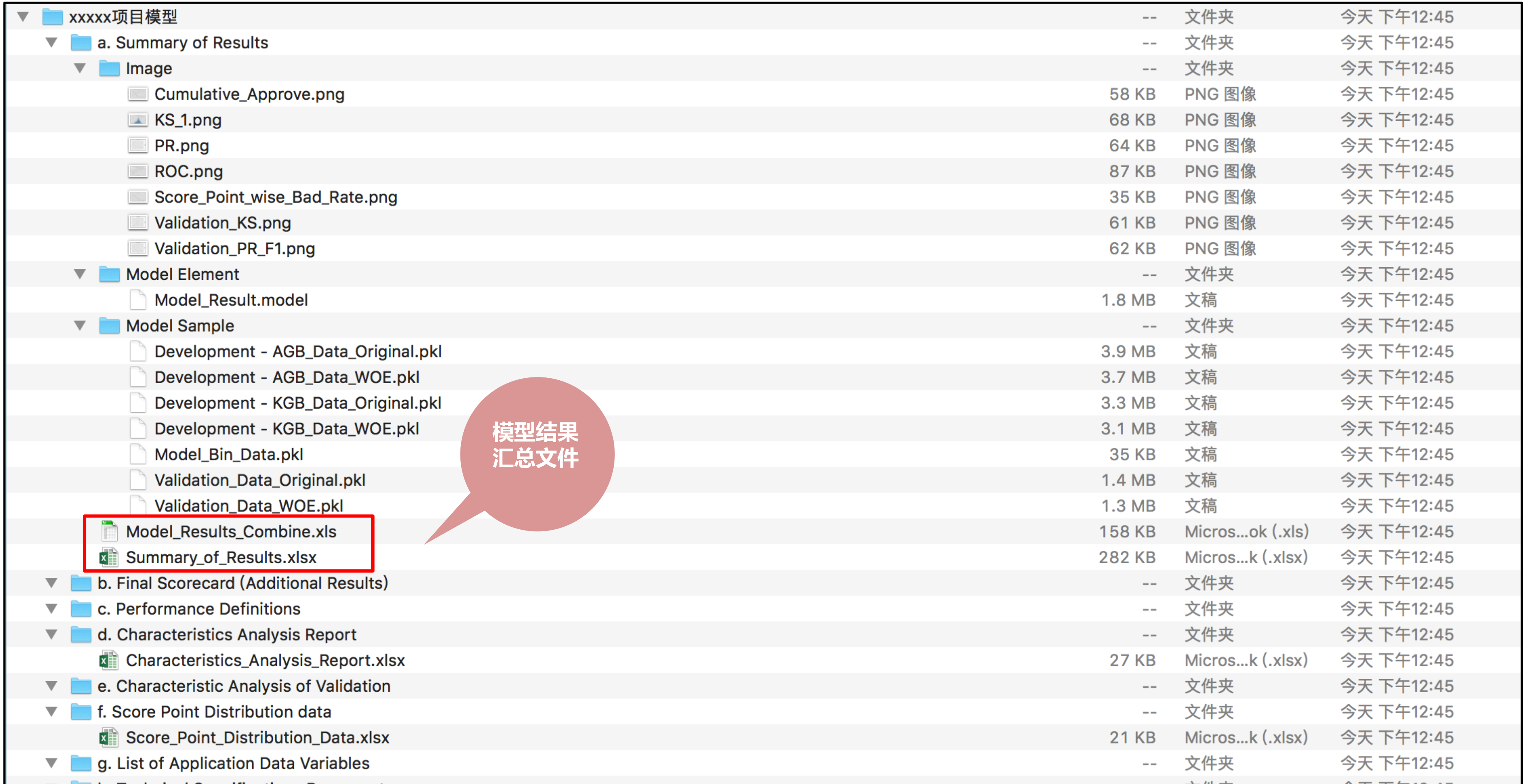
code demo

模型结果
标准化输出

模型结果标准化输出

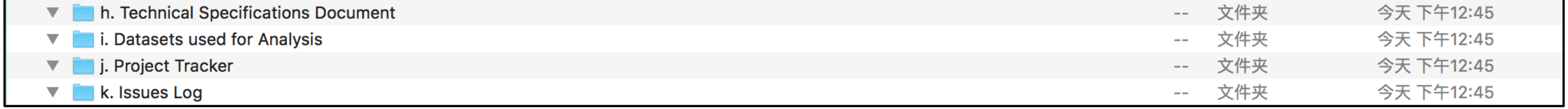
（1）将所有结果自动输出为一个文件夹
（2）那这个文件夹长什么样子呢？

模型结果标准化输出



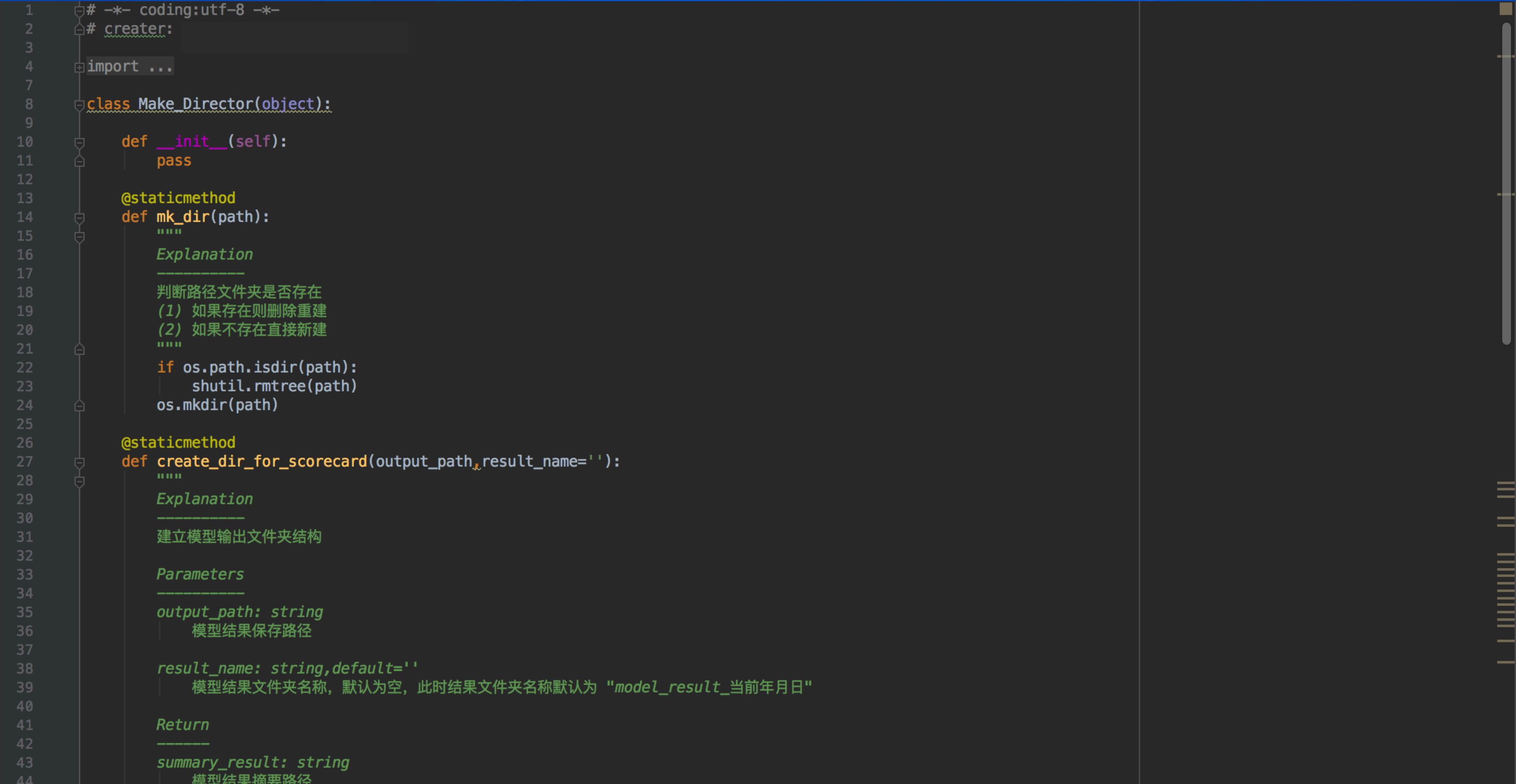
```
▼ xxxxx项目模型                                              --       文件夹            今天 下午12:45
  ▼ a. Summary of Results                                    --       文件夹            今天 下午12:45
    ▼ Image                                                  --       文件夹            今天 下午12:45
        Cumulative_Approve.png                            58 KB       PNG 图像          今天 下午12:45
        KS_1.png                                          68 KB       PNG 图像          今天 下午12:45
        PR.png                                            64 KB       PNG 图像          今天 下午12:45
        ROC.png                                           87 KB       PNG 图像          今天 下午12:45
        Score_Point_wise_Bad_Rate.png                     35 KB       PNG 图像          今天 下午12:45
        Validation_KS.png                                 61 KB       PNG 图像          今天 下午12:45
        Validation_PR_F1.png                              62 KB       PNG 图像          今天 下午12:45
    ▼ Model Element                                          --       文件夹            今天 下午12:45
        Model_Result.model                                1.8 MB      文稿              今天 下午12:45
    ▼ Model Sample                                           --       文件夹            今天 下午12:45
        Development - AGB_Data_Original.pkl               3.9 MB      文稿              今天 下午12:45
        Development - AGB_Data_WOE.pkl                    3.7 MB      文稿              今天 下午12:45
        Development - KGB_Data_Original.pkl               3.3 MB      文稿              今天 下午12:45
        Development - KGB_Data_WOE.pkl                    3.1 MB      文稿              今天 下午12:45
        Model_Bin_Data.pkl                                35 KB       文稿              今天 下午12:45
        Validation_Data_Original.pkl                      1.4 MB      文稿              今天 下午12:45
        Validation_Data_WOE.pkl                           1.3 MB      文稿              今天 下午12:45
      Model_Results_Combine.xls                           158 KB      Micros...ok (.xls)  今天 下午12:45
      Summary_of_Results.xlsx                             282 KB      Micros...k (.xlsx)  今天 下午12:45
  ▼ b. Final Scorecard (Additional Results)                  --       文件夹            今天 下午12:45
  ▼ c. Performance Definitions                               --       文件夹            今天 下午12:45
  ▼ d. Characteristics Analysis Report                       --       文件夹            今天 下午12:45
      Characteristics_Analysis_Report.xlsx                27 KB       Micros...k (.xlsx)  今天 下午12:45
  ▼ e. Characteristic Analysis of Validation                 --       文件夹            今天 下午12:45
  ▼ f. Score Point Distribution data                         --       文件夹            今天 下午12:45
      Score_Point_Distribution_Data.xlsx                  21 KB       Micros...k (.xlsx)  今天 下午12:45
  ▼ g. List of Application Data Variables                    --       文件夹            今天 下午12:45
```

模型结果
汇总文件

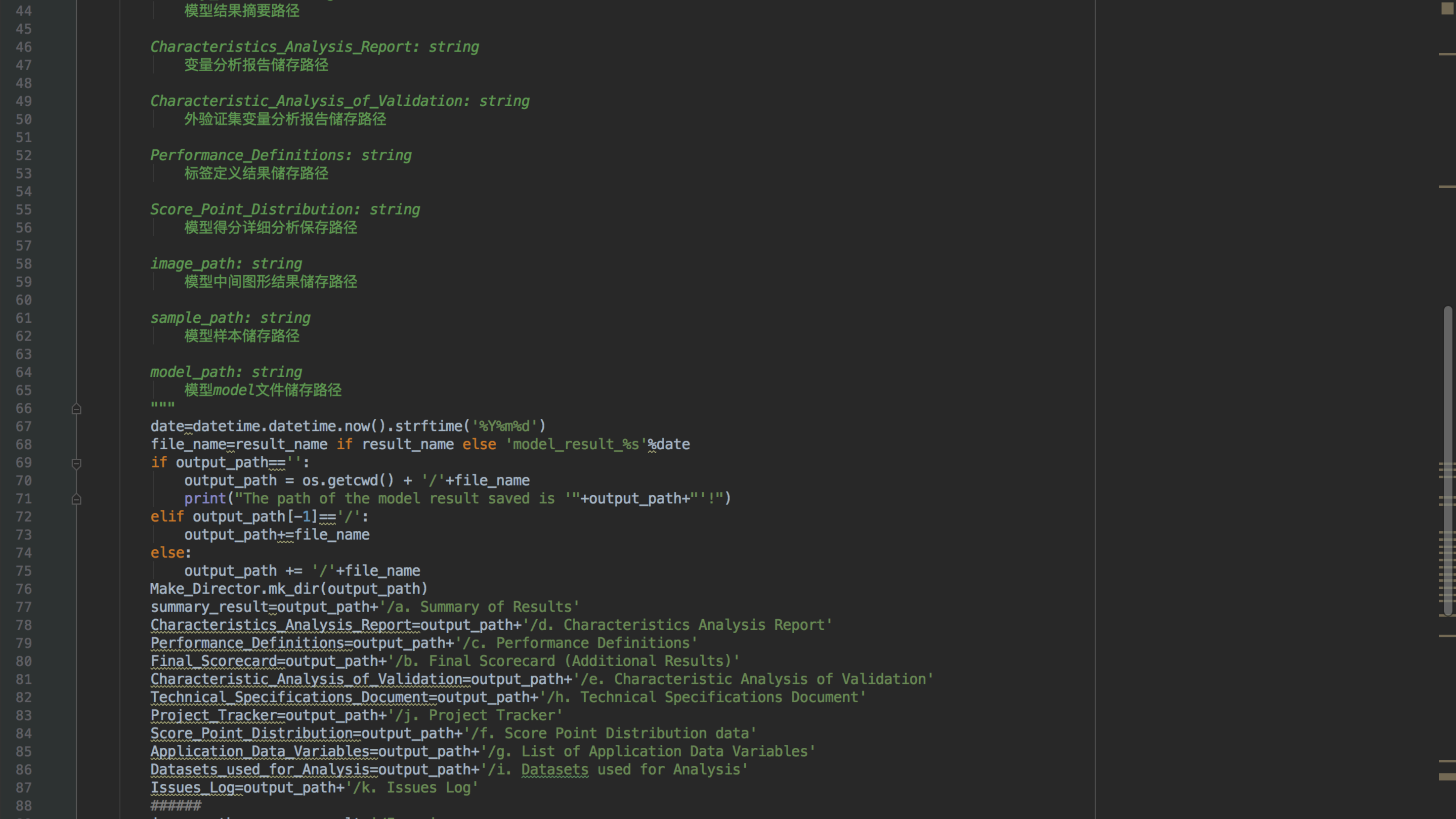


▼ h. Technical Specifications Document	--	文件夹	今天 下午12:45
▼ i. Datasets used for Analysis	--	文件夹	今天 下午12:45
▼ j. Project Tracker	--	文件夹	今天 下午12:45
▼ k. Issues Log	--	文件夹	今天 下午12:45

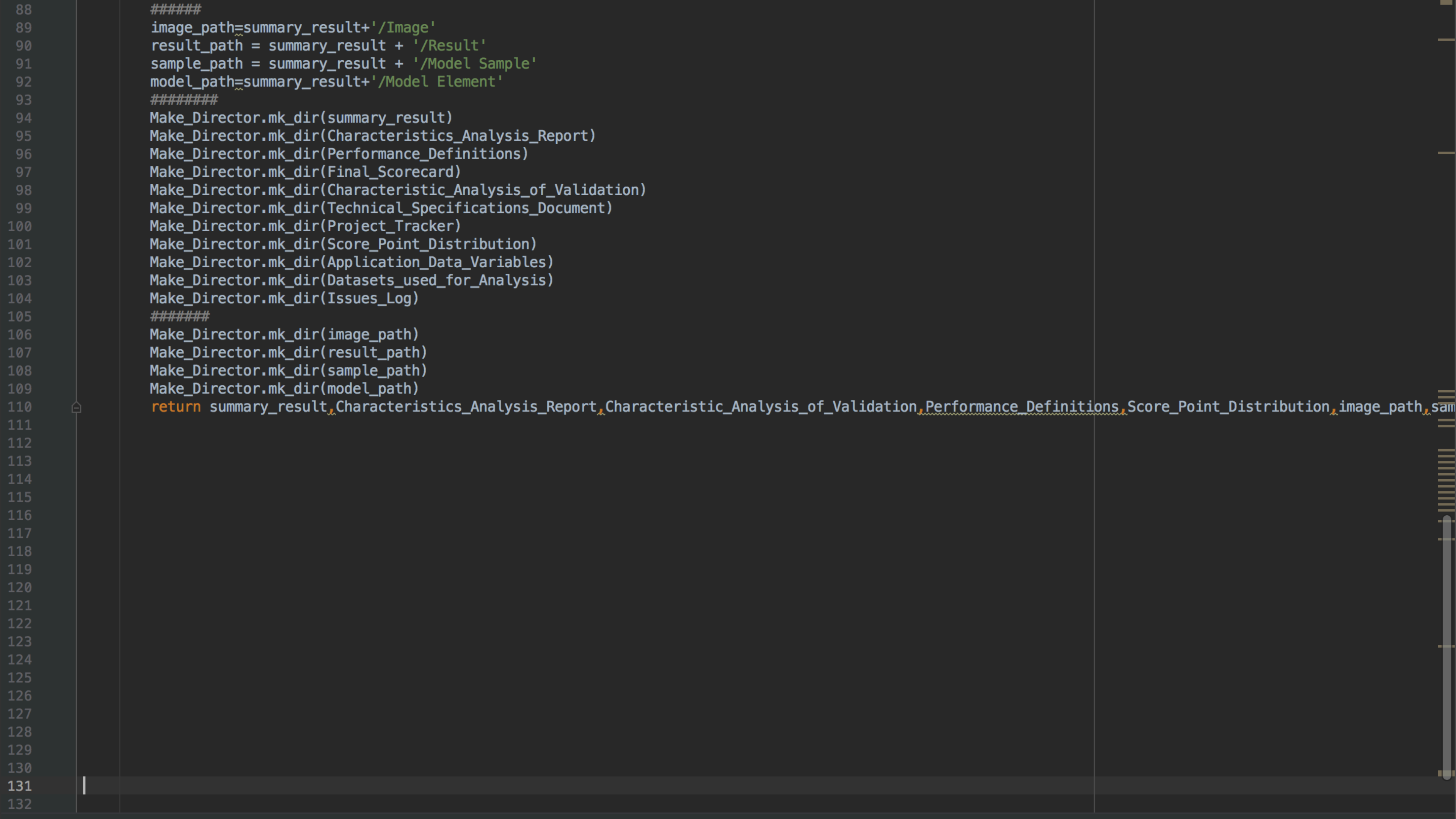
1 创建模型输出结构



```python
# -*- coding:utf-8 -*-
# creater:

import ...

class Make_Director(object):

    def __init__(self):
        pass

    @staticmethod
    def mk_dir(path):
        """
        Explanation
        -----------
        判断路径文件夹是否存在
        (1) 如果存在则删除重建
        (2) 如果不存在直接新建
        """
        if os.path.isdir(path):
            shutil.rmtree(path)
        os.mkdir(path)

    @staticmethod
    def create_dir_for_scorecard(output_path,result_name=''):
        """
        Explanation
        -----------
        建立模型输出文件夹结构

        Parameters
        -----------
        output_path: string
            模型结果保存路径

        result_name: string,default=''
            模型结果文件夹名称，默认为空，此时结果文件夹名称默认为 "model_result_当前年月日"

        Return
        -------
        summary_result: string
            模型结果摘要路径
```

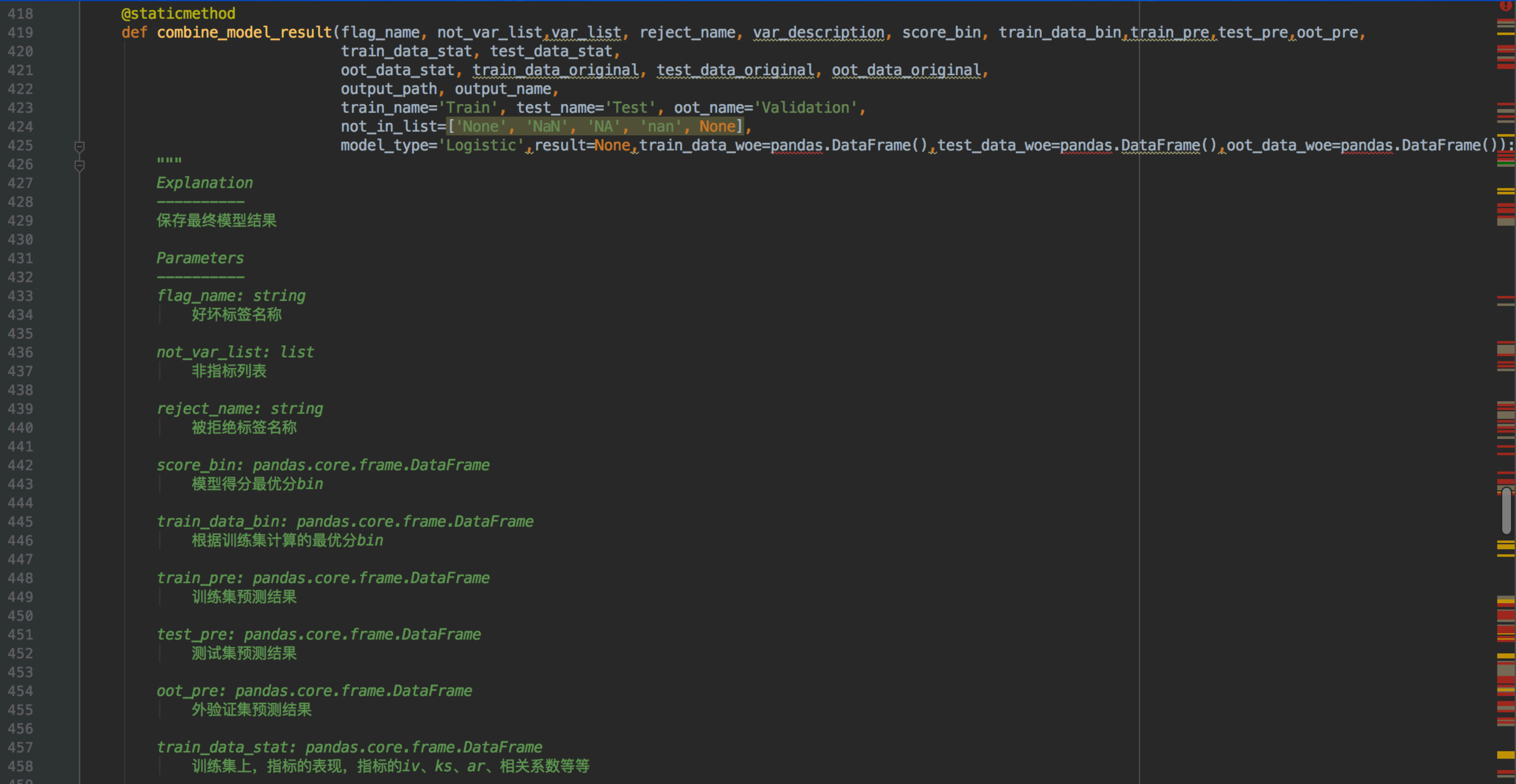


```
        模型结果摘要路径

    Characteristics_Analysis_Report: string
        变量分析报告储存路径

    Characteristic_Analysis_of_Validation: string
        外验证集变量分析报告储存路径

    Performance_Definitions: string
        标签定义结果储存路径

    Score_Point_Distribution: string
        模型得分详细分析保存路径

    image_path: string
        模型中间图形结果储存路径

    sample_path: string
        模型样本储存路径

    model_path: string
        模型model文件储存路径
    """
    date=datetime.datetime.now().strftime('%Y%m%d')
    file_name=result_name if result_name else 'model_result_%s'%date
    if output_path=='':
        output_path = os.getcwd() + '/'+file_name
        print("The path of the model result saved is '"+output_path+"'!")
    elif output_path[-1]=='/':
        output_path+=file_name
    else:
        output_path += '/'+file_name
    Make_Director.mk_dir(output_path)
    summary_result=output_path+'/a. Summary of Results'
    Characteristics_Analysis_Report=output_path+'/d. Characteristics Analysis Report'
    Performance_Definitions=output_path+'/c. Performance Definitions'
    Final_Scorecard=output_path+'/b. Final Scorecard (Additional Results)'
    Characteristic_Analysis_of_Validation=output_path+'/e. Characteristic Analysis of Validation'
    Technical_Specifications_Document=output_path+'/h. Technical Specifications Document'
    Project_Tracker=output_path+'/j. Project Tracker'
    Score_Point_Distribution=output_path+'/f. Score Point Distribution data'
    Application_Data_Variables=output_path+'/g. List of Application Data Variables'
    Datasets_used_for_Analysis=output_path+'/i. Datasets used for Analysis'
    Issues_Log=output_path+'/k. Issues Log'
    ######
```

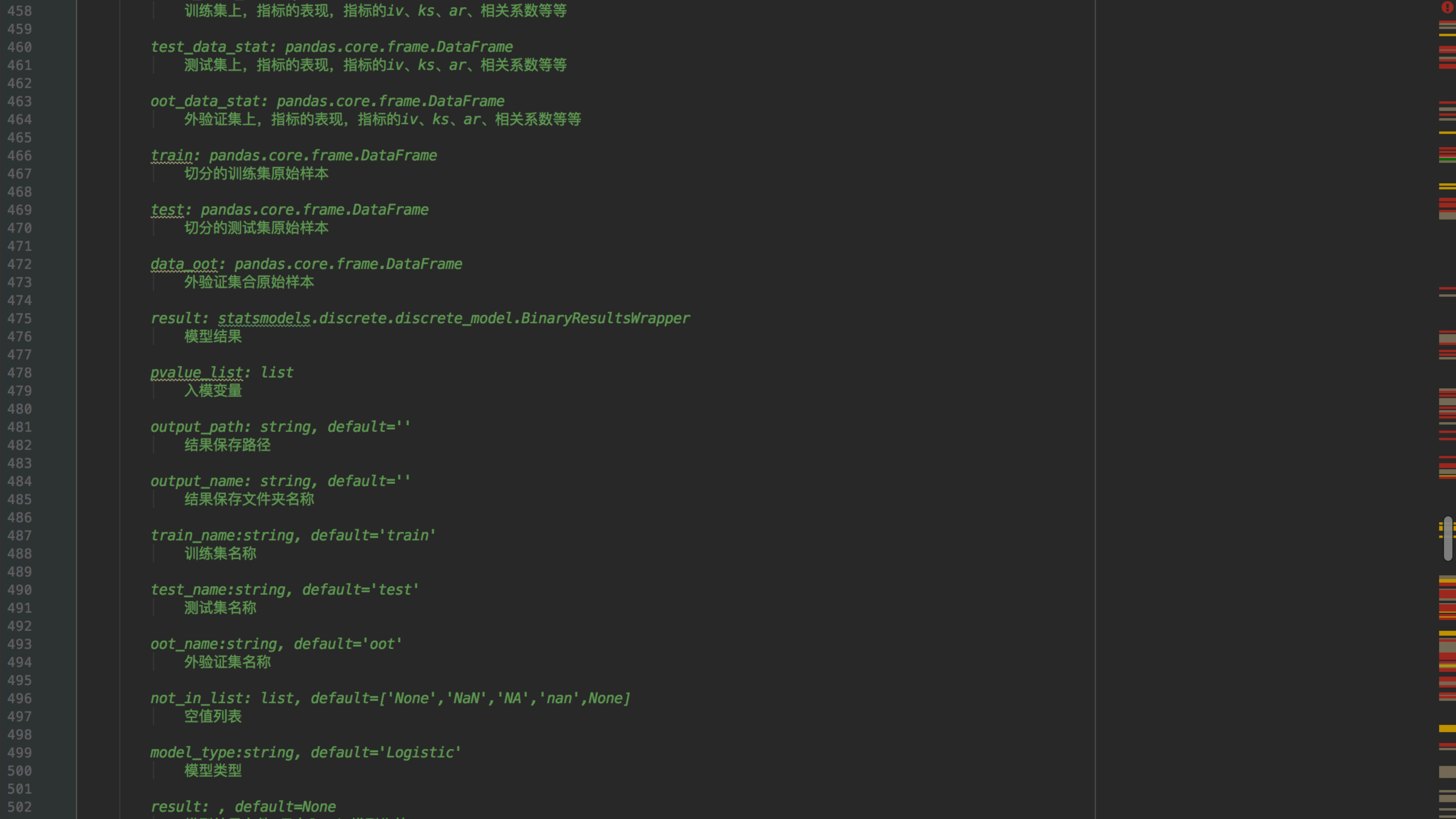


```
        ######
        image_path=summary_result+'/Image'
        result_path = summary_result + '/Result'
        sample_path = summary_result + '/Model Sample'
        model_path=summary_result+'/Model Element'
        ########
        Make_Director.mk_dir(summary_result)
        Make_Director.mk_dir(Characteristics_Analysis_Report)
        Make_Director.mk_dir(Performance_Definitions)
        Make_Director.mk_dir(Final_Scorecard)
        Make_Director.mk_dir(Characteristic_Analysis_of_Validation)
        Make_Director.mk_dir(Technical_Specifications_Document)
        Make_Director.mk_dir(Project_Tracker)
        Make_Director.mk_dir(Score_Point_Distribution)
        Make_Director.mk_dir(Application_Data_Variables)
        Make_Director.mk_dir(Datasets_used_for_Analysis)
        Make_Director.mk_dir(Issues_Log)
        #######
        Make_Director.mk_dir(image_path)
        Make_Director.mk_dir(result_path)
        Make_Director.mk_dir(sample_path)
        Make_Director.mk_dir(model_path)
        return summary_result,Characteristics_Analysis_Report,Characteristic_Analysis_of_Validation,Performance_Definitions,Score_Point_Distribution,image_path,sam
```

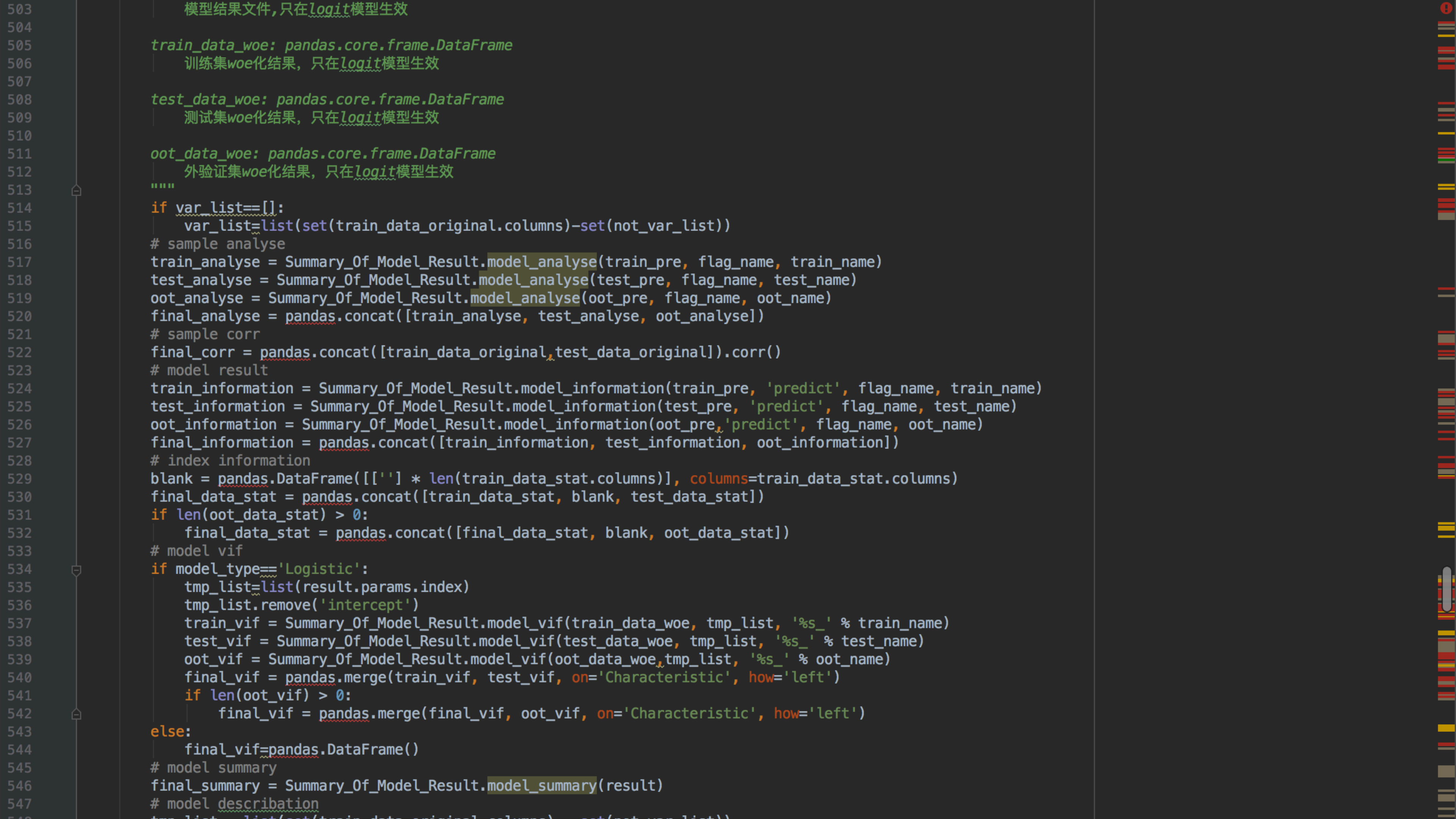
② 模型结果汇总输出



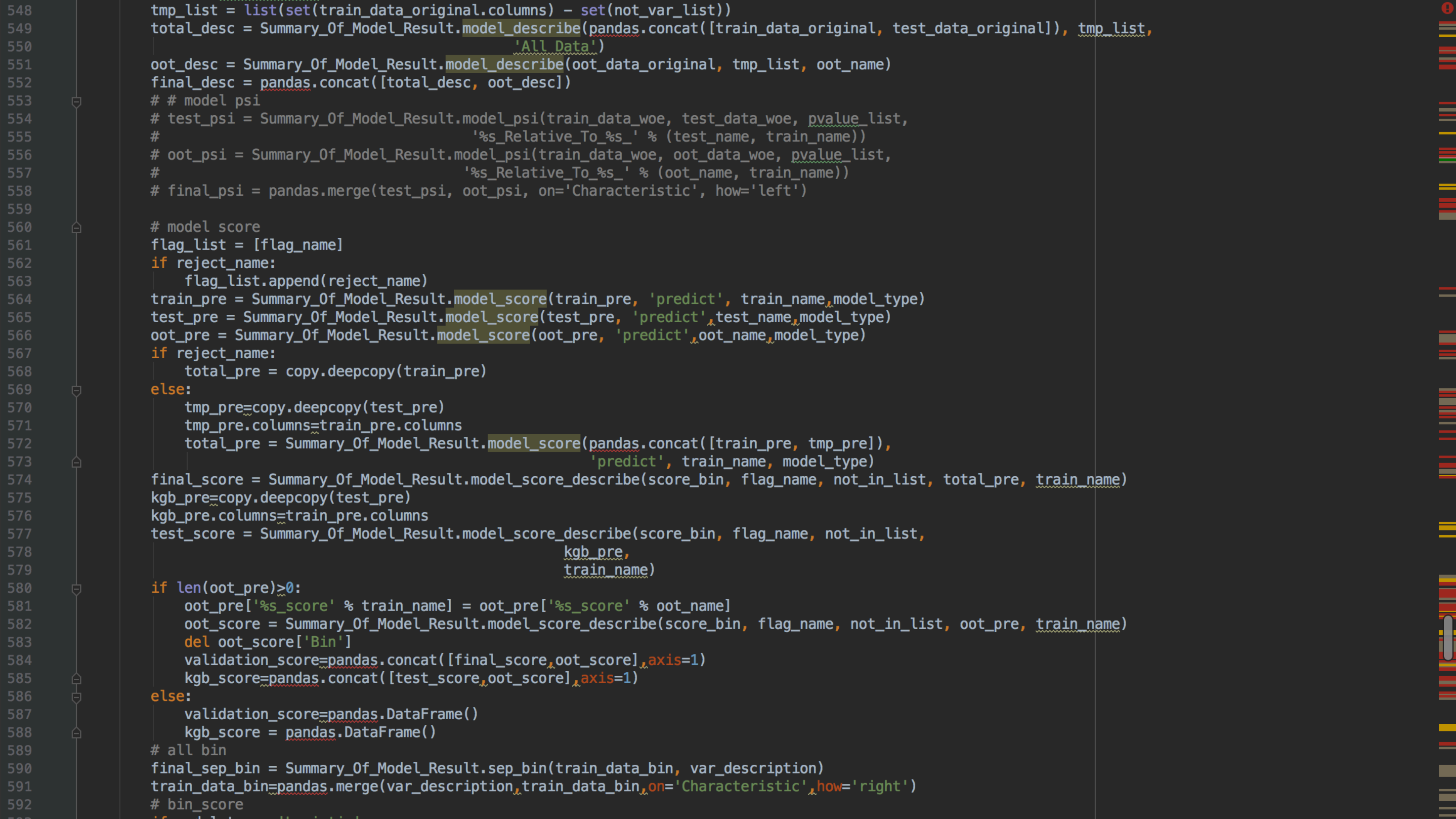
```python
    @staticmethod
    def combine_model_result(flag_name, not_var_list,var_list, reject_name, var_description, score_bin, train_data_bin,train_pre,test_pre,oot_pre,
                             train_data_stat, test_data_stat,
                             oot_data_stat, train_data_original, test_data_original, oot_data_original,
                             output_path, output_name,
                             train_name='Train', test_name='Test', oot_name='Validation',
                             not_in_list=['None', 'NaN', 'NA', 'nan', None],
                             model_type='Logistic',result=None,train_data_woe=pandas.DataFrame(),test_data_woe=pandas.DataFrame(),oot_data_woe=pandas.DataFrame()):
        """
        Explanation
        ----------
        保存最终模型结果

        Parameters
        ----------
        flag_name: string
            好坏标签名称

        not_var_list: list
            非指标列表

        reject_name: string
            被拒绝标签名称

        score_bin: pandas.core.frame.DataFrame
            模型得分最优分bin

        train_data_bin: pandas.core.frame.DataFrame
            根据训练集计算的最优分bin

        train_pre: pandas.core.frame.DataFrame
            训练集预测结果

        test_pre: pandas.core.frame.DataFrame
            测试集预测结果

        oot_pre: pandas.core.frame.DataFrame
            外验证集预测结果

        train_data_stat: pandas.core.frame.DataFrame
            训练集上，指标的表现，指标的iv、ks、ar、相关系数等等
```



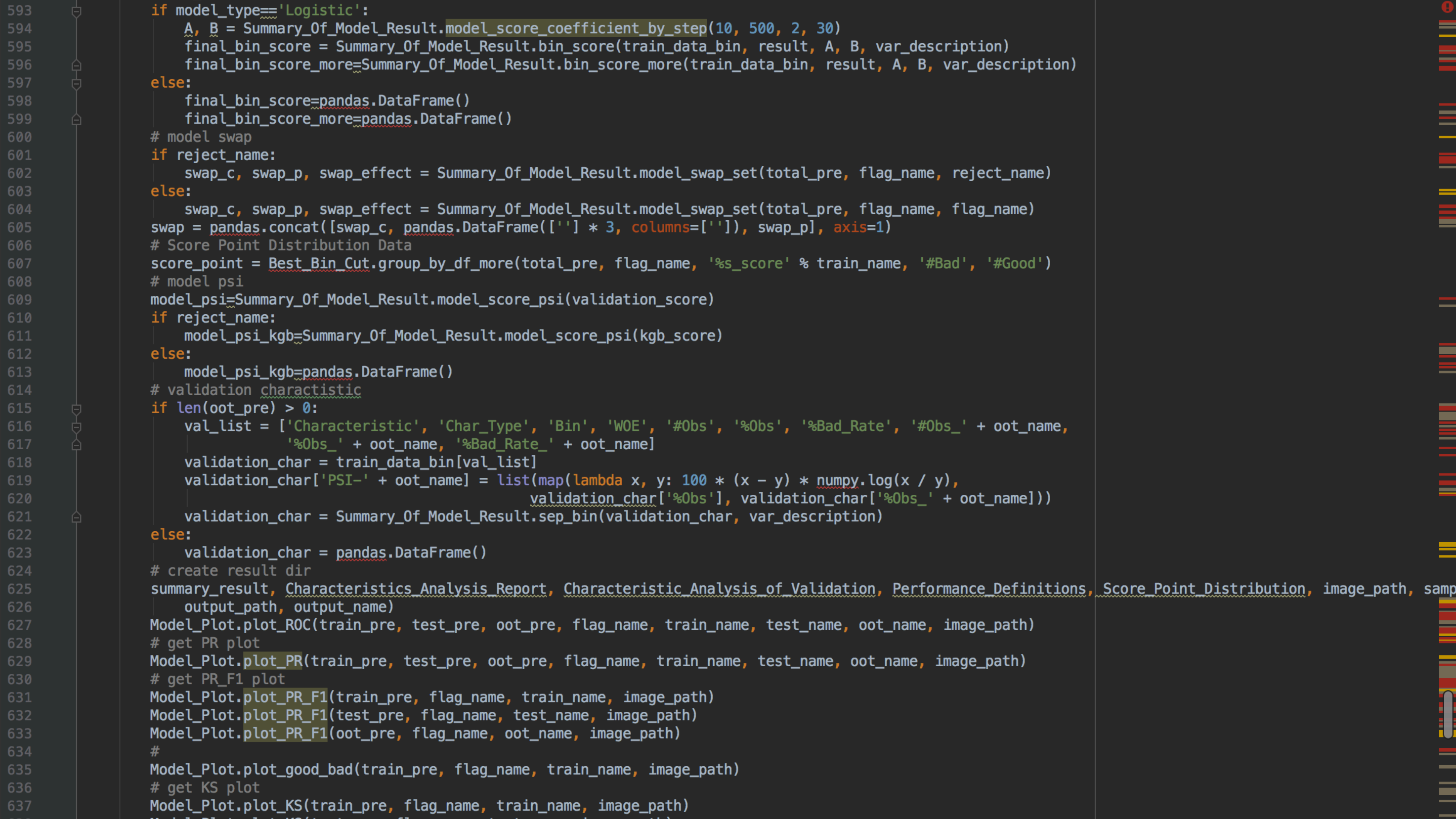
```
    训练集上，指标的表现，指标的iv、ks、ar、相关系数等等

test_data_stat: pandas.core.frame.DataFrame
    测试集上，指标的表现，指标的iv、ks、ar、相关系数等等

oot_data_stat: pandas.core.frame.DataFrame
    外验证集上，指标的表现，指标的iv、ks、ar、相关系数等等

train: pandas.core.frame.DataFrame
    切分的训练集原始样本

test: pandas.core.frame.DataFrame
    切分的测试集原始样本

data_oot: pandas.core.frame.DataFrame
    外验证集合原始样本

result: statsmodels.discrete.discrete_model.BinaryResultsWrapper
    模型结果

pvalue_list: list
    入模变量

output_path: string, default=''
    结果保存路径

output_name: string, default=''
    结果保存文件夹名称

train_name:string, default='train'
    训练集名称

test_name:string, default='test'
    测试集名称

oot_name:string, default='oot'
    外验证集名称

not_in_list: list, default=['None','NaN','NA','nan',None]
    空值列表

model_type:string, default='Logistic'
    模型类型

result: , default=None
```



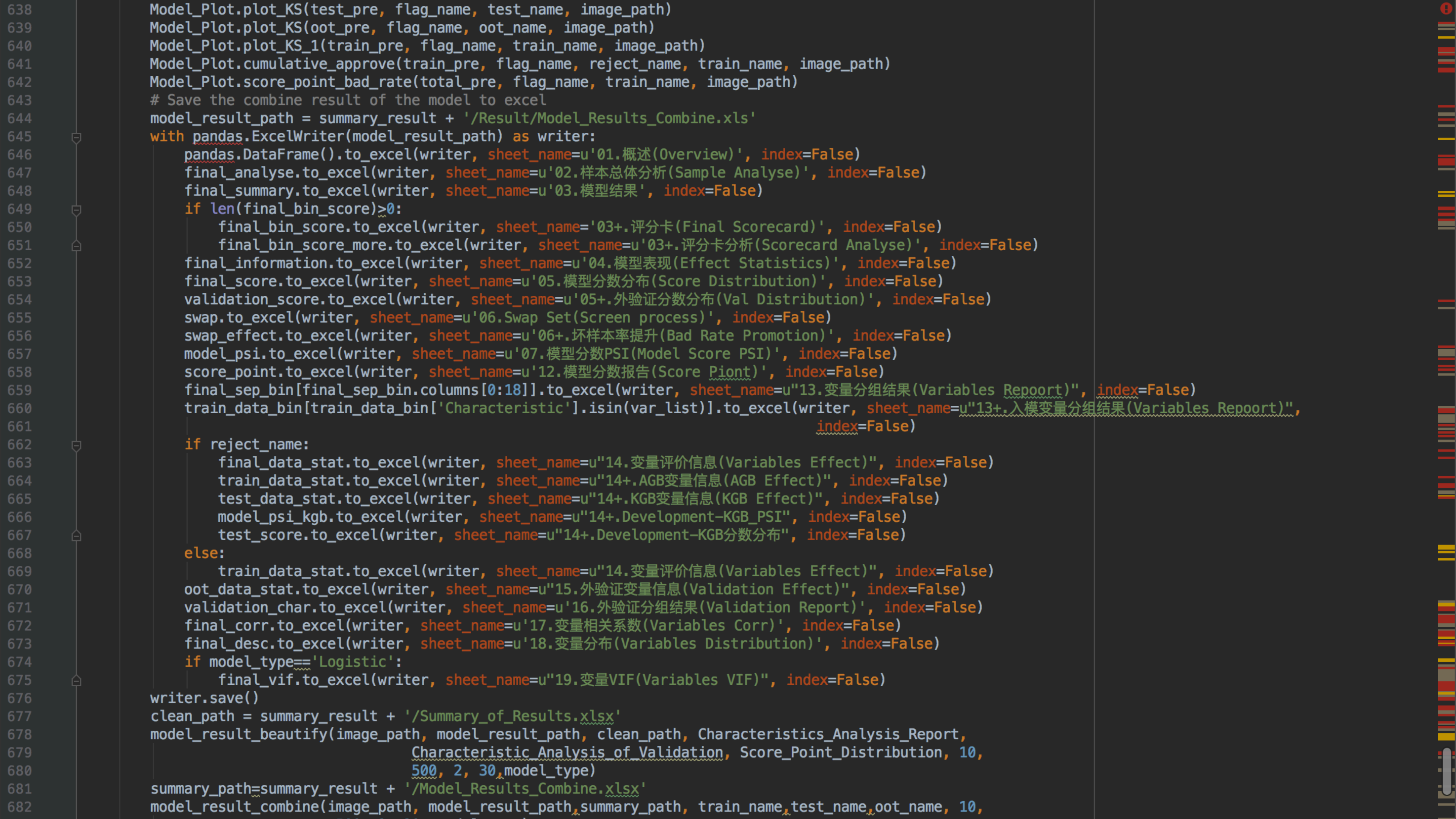
```
        模型结果文件,只在logit模型生效

    train_data_woe: pandas.core.frame.DataFrame
        训练集woe化结果，只在logit模型生效

    test_data_woe: pandas.core.frame.DataFrame
        测试集woe化结果，只在logit模型生效

    oot_data_woe: pandas.core.frame.DataFrame
        外验证集woe化结果，只在logit模型生效
    """
    if var_list==[]:
        var_list=list(set(train_data_original.columns)-set(not_var_list))
    # sample analyse
    train_analyse = Summary_Of_Model_Result.model_analyse(train_pre, flag_name, train_name)
    test_analyse = Summary_Of_Model_Result.model_analyse(test_pre, flag_name, test_name)
    oot_analyse = Summary_Of_Model_Result.model_analyse(oot_pre, flag_name, oot_name)
    final_analyse = pandas.concat([train_analyse, test_analyse, oot_analyse])
    # sample corr
    final_corr = pandas.concat([train_data_original,test_data_original]).corr()
    # model result
    train_information = Summary_Of_Model_Result.model_information(train_pre, 'predict', flag_name, train_name)
    test_information = Summary_Of_Model_Result.model_information(test_pre, 'predict', flag_name, test_name)
    oot_information = Summary_Of_Model_Result.model_information(oot_pre,'predict', flag_name, oot_name)
    final_information = pandas.concat([train_information, test_information, oot_information])
    # index information
    blank = pandas.DataFrame([[''] * len(train_data_stat.columns)], columns=train_data_stat.columns)
    final_data_stat = pandas.concat([train_data_stat, blank, test_data_stat])
    if len(oot_data_stat) > 0:
        final_data_stat = pandas.concat([final_data_stat, blank, oot_data_stat])
    # model vif
    if model_type=='Logistic':
        tmp_list=list(result.params.index)
        tmp_list.remove('intercept')
        train_vif = Summary_Of_Model_Result.model_vif(train_data_woe, tmp_list, '%s_' % train_name)
        test_vif = Summary_Of_Model_Result.model_vif(test_data_woe, tmp_list, '%s_' % test_name)
        oot_vif = Summary_Of_Model_Result.model_vif(oot_data_woe,tmp_list, '%s_' % oot_name)
        final_vif = pandas.merge(train_vif, test_vif, on='Characteristic', how='left')
        if len(oot_vif) > 0:
            final_vif = pandas.merge(final_vif, oot_vif, on='Characteristic', how='left')
    else:
        final_vif=pandas.DataFrame()
    # model summary
    final_summary = Summary_Of_Model_Result.model_summary(result)
    # model description
    tmp_list = list(set(train_data_original.columns) - set(not_var_list))
```



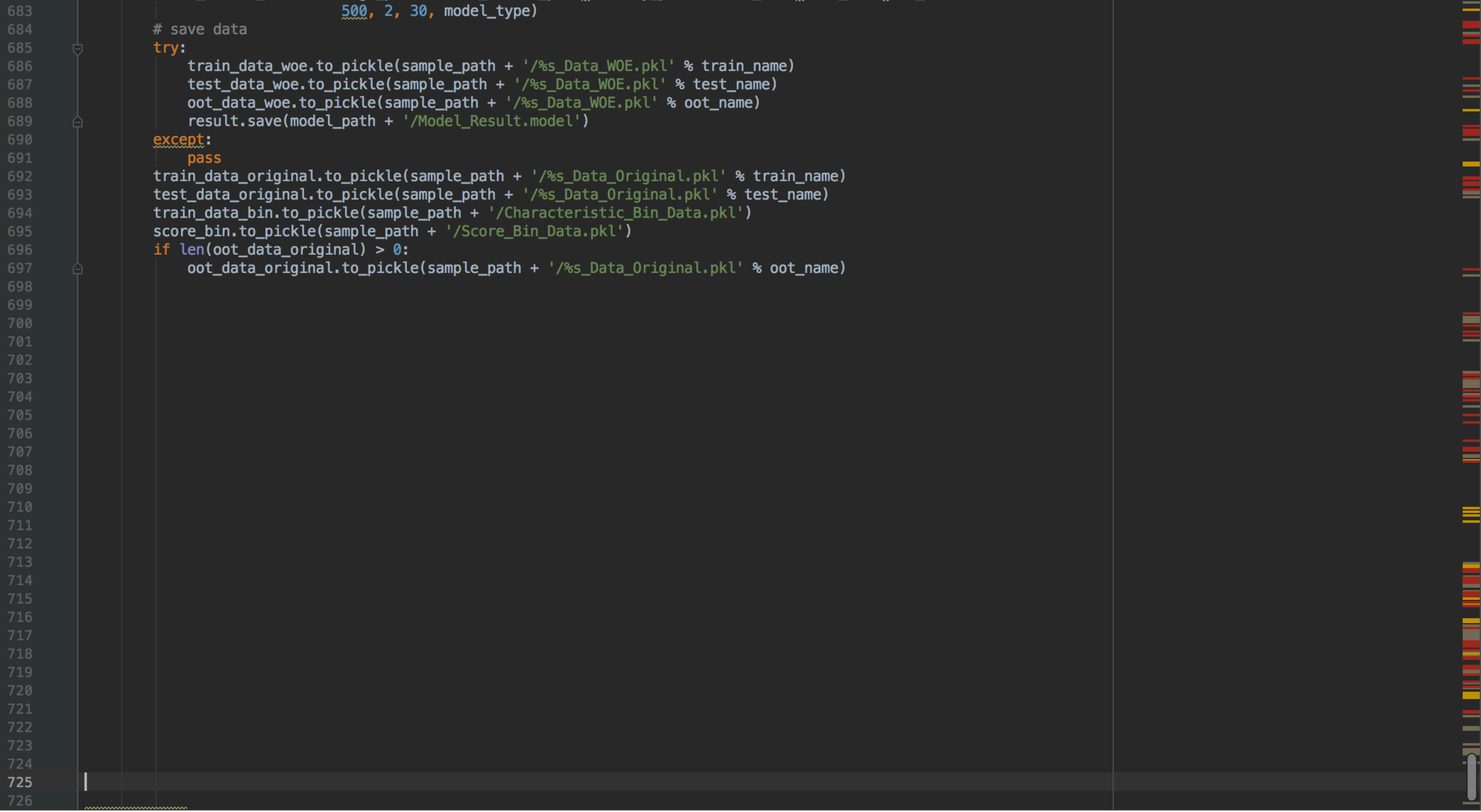
```
        tmp_list = list(set(train_data_original.columns) - set(not_var_list))
        total_desc = Summary_Of_Model_Result.model_describe(pandas.concat([train_data_original, test_data_original]), tmp_list,
                                           'All Data')
        oot_desc = Summary_Of_Model_Result.model_describe(oot_data_original, tmp_list, oot_name)
        final_desc = pandas.concat([total_desc, oot_desc])
        # # model psi
        # test_psi = Summary_Of_Model_Result.model_psi(train_data_woe, test_data_woe, pvalue_list,
        #                                       '%s_Relative_To_%s_' % (test_name, train_name))
        # oot_psi = Summary_Of_Model_Result.model_psi(train_data_woe, oot_data_woe, pvalue_list,
        #                                      '%s_Relative_To_%s_' % (oot_name, train_name))
        # final_psi = pandas.merge(test_psi, oot_psi, on='Characteristic', how='left')

        # model score
        flag_list = [flag_name]
        if reject_name:
            flag_list.append(reject_name)
        train_pre = Summary_Of_Model_Result.model_score(train_pre, 'predict', train_name,model_type)
        test_pre = Summary_Of_Model_Result.model_score(test_pre, 'predict',test_name,model_type)
        oot_pre = Summary_Of_Model_Result.model_score(oot_pre, 'predict',oot_name,model_type)
        if reject_name:
            total_pre = copy.deepcopy(train_pre)
        else:
            tmp_pre=copy.deepcopy(test_pre)
            tmp_pre.columns=train_pre.columns
            total_pre = Summary_Of_Model_Result.model_score(pandas.concat([train_pre, tmp_pre]),
                                                    'predict', train_name, model_type)
        final_score = Summary_Of_Model_Result.model_score_describe(score_bin, flag_name, not_in_list, total_pre, train_name)
        kgb_pre=copy.deepcopy(test_pre)
        kgb_pre.columns=train_pre.columns
        test_score = Summary_Of_Model_Result.model_score_describe(score_bin, flag_name, not_in_list,
                                                 kgb_pre,
                                                 train_name)
        if len(oot_pre)>0:
            oot_pre['%s_score' % train_name] = oot_pre['%s_score' % oot_name]
            oot_score = Summary_Of_Model_Result.model_score_describe(score_bin, flag_name, not_in_list, oot_pre, train_name)
            del oot_score['Bin']
            validation_score=pandas.concat([final_score,oot_score],axis=1)
            kgb_score=pandas.concat([test_score,oot_score],axis=1)
        else:
            validation_score=pandas.DataFrame()
            kgb_score = pandas.DataFrame()
        # all bin
        final_sep_bin = Summary_Of_Model_Result.sep_bin(train_data_bin, var_description)
        train_data_bin=pandas.merge(var_description,train_data_bin,on='Characteristic',how='right')
        # bin_score
```



```
        if model_type=='Logistic':
            A, B = Summary_Of_Model_Result.model_score_coefficient_by_step(10, 500, 2, 30)
            final_bin_score = Summary_Of_Model_Result.bin_score(train_data_bin, result, A, B, var_description)
            final_bin_score_more=Summary_Of_Model_Result.bin_score_more(train_data_bin, result, A, B, var_description)
        else:
            final_bin_score=pandas.DataFrame()
            final_bin_score_more=pandas.DataFrame()
        # model swap
        if reject_name:
            swap_c, swap_p, swap_effect = Summary_Of_Model_Result.model_swap_set(total_pre, flag_name, reject_name)
        else:
            swap_c, swap_p, swap_effect = Summary_Of_Model_Result.model_swap_set(total_pre, flag_name, flag_name)
        swap = pandas.concat([swap_c, pandas.DataFrame([''] * 3, columns=['']), swap_p], axis=1)
        # Score Point Distribution Data
        score_point = Best_Bin_Cut.group_by_df_more(total_pre, flag_name, '%s_score' % train_name, '#Bad', '#Good')
        # model psi
        model_psi=Summary_Of_Model_Result.model_score_psi(validation_score)
        if reject_name:
            model_psi_kgb=Summary_Of_Model_Result.model_score_psi(kgb_score)
        else:
            model_psi_kgb=pandas.DataFrame()
        # validation charactistic
        if len(oot_pre) > 0:
            val_list = ['Characteristic', 'Char_Type', 'Bin', 'WOE', '#Obs', '%Obs', '%Bad_Rate', '#Obs_' + oot_name,
                        '%Obs_' + oot_name, '%Bad_Rate_' + oot_name]
            validation_char = train_data_bin[val_list]
            validation_char['PSI-' + oot_name] = list(map(lambda x, y: 100 * (x - y) * numpy.log(x / y),
                                                     validation_char['%Obs'], validation_char['%Obs_' + oot_name]))
            validation_char = Summary_Of_Model_Result.sep_bin(validation_char, var_description)
        else:
            validation_char = pandas.DataFrame()
        # create result dir
        summary_result, Characteristics_Analysis_Report, Characteristic_Analysis_of_Validation, Performance_Definitions, Score_Point_Distribution, image_path, samp
            output_path, output_name)
        Model_Plot.plot_ROC(train_pre, test_pre, oot_pre, flag_name, train_name, test_name, oot_name, image_path)
        # get PR plot
        Model_Plot.plot_PR(train_pre, test_pre, oot_pre, flag_name, train_name, test_name, oot_name, image_path)
        # get PR_F1 plot
        Model_Plot.plot_PR_F1(train_pre, flag_name, train_name, image_path)
        Model_Plot.plot_PR_F1(test_pre, flag_name, test_name, image_path)
        Model_Plot.plot_PR_F1(oot_pre, flag_name, oot_name, image_path)
        #
        Model_Plot.plot_good_bad(train_pre, flag_name, train_name, image_path)
        # get KS plot
        Model_Plot.plot_KS(train_pre, flag_name, train_name, image_path)
```

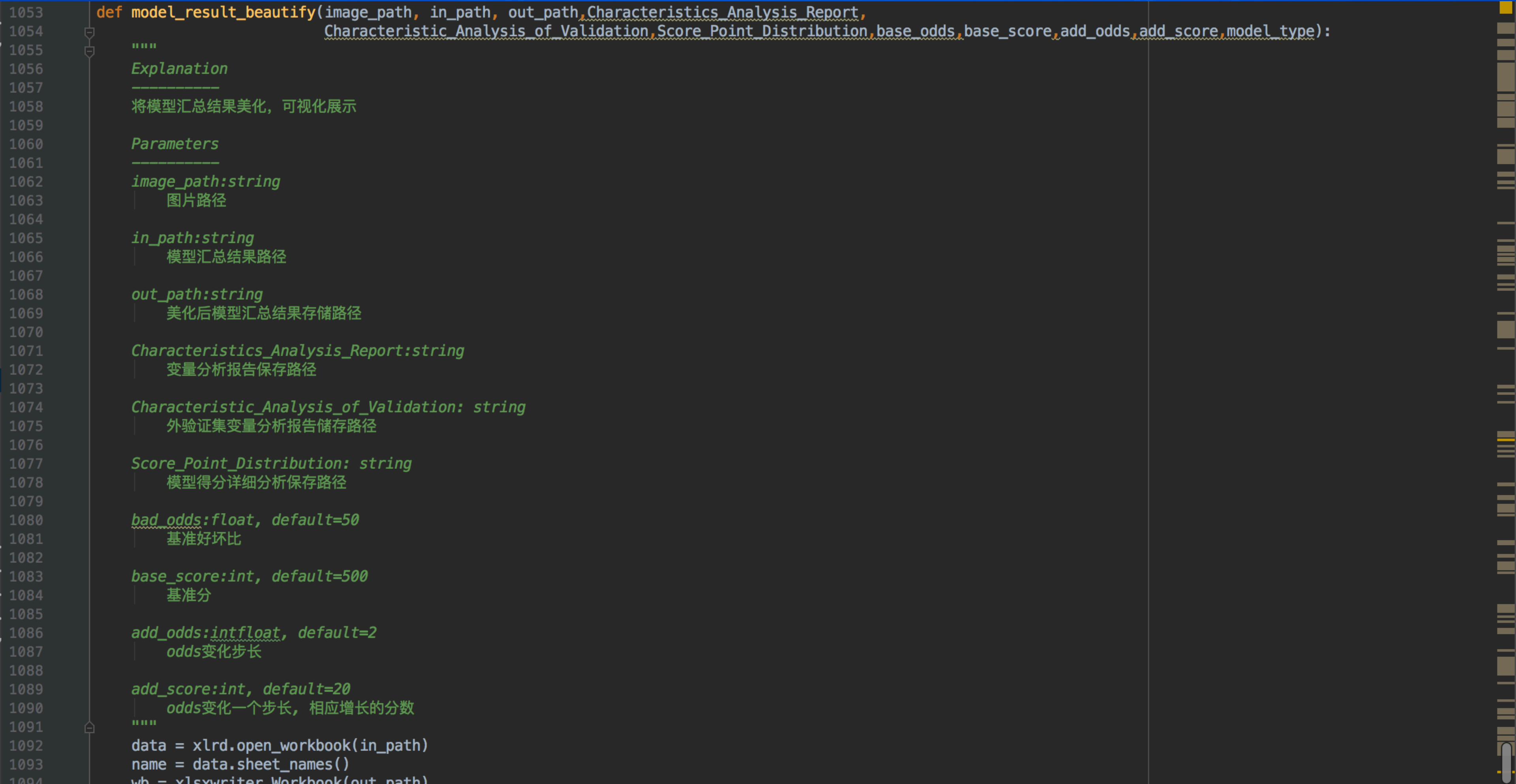


```
        Model_Plot.plot_KS(test_pre, flag_name, test_name, image_path)
        Model_Plot.plot_KS(oot_pre, flag_name, oot_name, image_path)
        Model_Plot.plot_KS_1(train_pre, flag_name, train_name, image_path)
        Model_Plot.cumulative_approve(train_pre, flag_name, reject_name, train_name, image_path)
        Model_Plot.score_point_bad_rate(total_pre, flag_name, train_name, image_path)
        # Save the combine result of the model to excel
        model_result_path = summary_result + '/Result/Model_Results_Combine.xls'
        with pandas.ExcelWriter(model_result_path) as writer:
            pandas.DataFrame().to_excel(writer, sheet_name=u'01.概述(Overview)', index=False)
            final_analyse.to_excel(writer, sheet_name=u'02.样本总体分析(Sample Analyse)', index=False)
            final_summary.to_excel(writer, sheet_name=u'03.模型结果', index=False)
            if len(final_bin_score)>0:
                final_bin_score.to_excel(writer, sheet_name='03+.评分卡(Final Scorecard)', index=False)
                final_bin_score_more.to_excel(writer, sheet_name=u'03+.评分卡分析(Scorecard Analyse)', index=False)
            final_information.to_excel(writer, sheet_name=u'04.模型表现(Effect Statistics)', index=False)
            final_score.to_excel(writer, sheet_name=u'05.模型分数分布(Score Distribution)', index=False)
            validation_score.to_excel(writer, sheet_name=u'05+.外验证分数分布(Val Distribution)', index=False)
            swap.to_excel(writer, sheet_name=u'06.Swap Set(Screen process)', index=False)
            swap_effect.to_excel(writer, sheet_name=u'06+.坏样本率提升(Bad Rate Promotion)', index=False)
            model_psi.to_excel(writer, sheet_name=u'07.模型分数PSI(Model Score PSI)', index=False)
            score_point.to_excel(writer, sheet_name=u'12.模型分数报告(Score Piont)', index=False)
            final_sep_bin[final_sep_bin.columns[0:18]].to_excel(writer, sheet_name=u"13.变量分组结果(Variables Repoort)", index=False)
            train_data_bin[train_data_bin['Characteristic'].isin(var_list)].to_excel(writer, sheet_name=u"13+.入模变量分组结果(Variables Repoort)",
                                                                                       index=False)
            if reject_name:
                final_data_stat.to_excel(writer, sheet_name=u"14.变量评价信息(Variables Effect)", index=False)
                train_data_stat.to_excel(writer, sheet_name=u"14+.AGB变量信息(AGB Effect)", index=False)
                test_data_stat.to_excel(writer, sheet_name=u"14+.KGB变量信息(KGB Effect)", index=False)
                model_psi_kgb.to_excel(writer, sheet_name=u"14+.Development-KGB_PSI", index=False)
                test_score.to_excel(writer, sheet_name=u"14+.Development-KGB分数分布", index=False)
            else:
                train_data_stat.to_excel(writer, sheet_name=u"14.变量评价信息(Variables Effect)", index=False)
            oot_data_stat.to_excel(writer, sheet_name=u"15.外验证变量信息(Validation Effect)", index=False)
            validation_char.to_excel(writer, sheet_name=u'16.外验证分组结果(Validation Report)', index=False)
            final_corr.to_excel(writer, sheet_name=u'17.变量相关系数(Variables Corr)', index=False)
            final_desc.to_excel(writer, sheet_name=u'18.变量分布(Variables Distribution)', index=False)
            if model_type=='Logistic':
                final_vif.to_excel(writer, sheet_name=u"19.变量VIF(Variables VIF)", index=False)
        writer.save()
        clean_path = summary_result + '/Summary_of_Results.xlsx'
        model_result_beautify(image_path, model_result_path, clean_path, Characteristics_Analysis_Report,
                              Characteristic_Analysis_of_Validation, Score_Point_Distribution, 10,
                              500, 2, 30,model_type)
        summary_path=summary_result + '/Model_Results_Combine.xlsx'
        model_result_combine(image_path, model_result_path,summary_path, train_name,test_name,oot_name, 10,
```

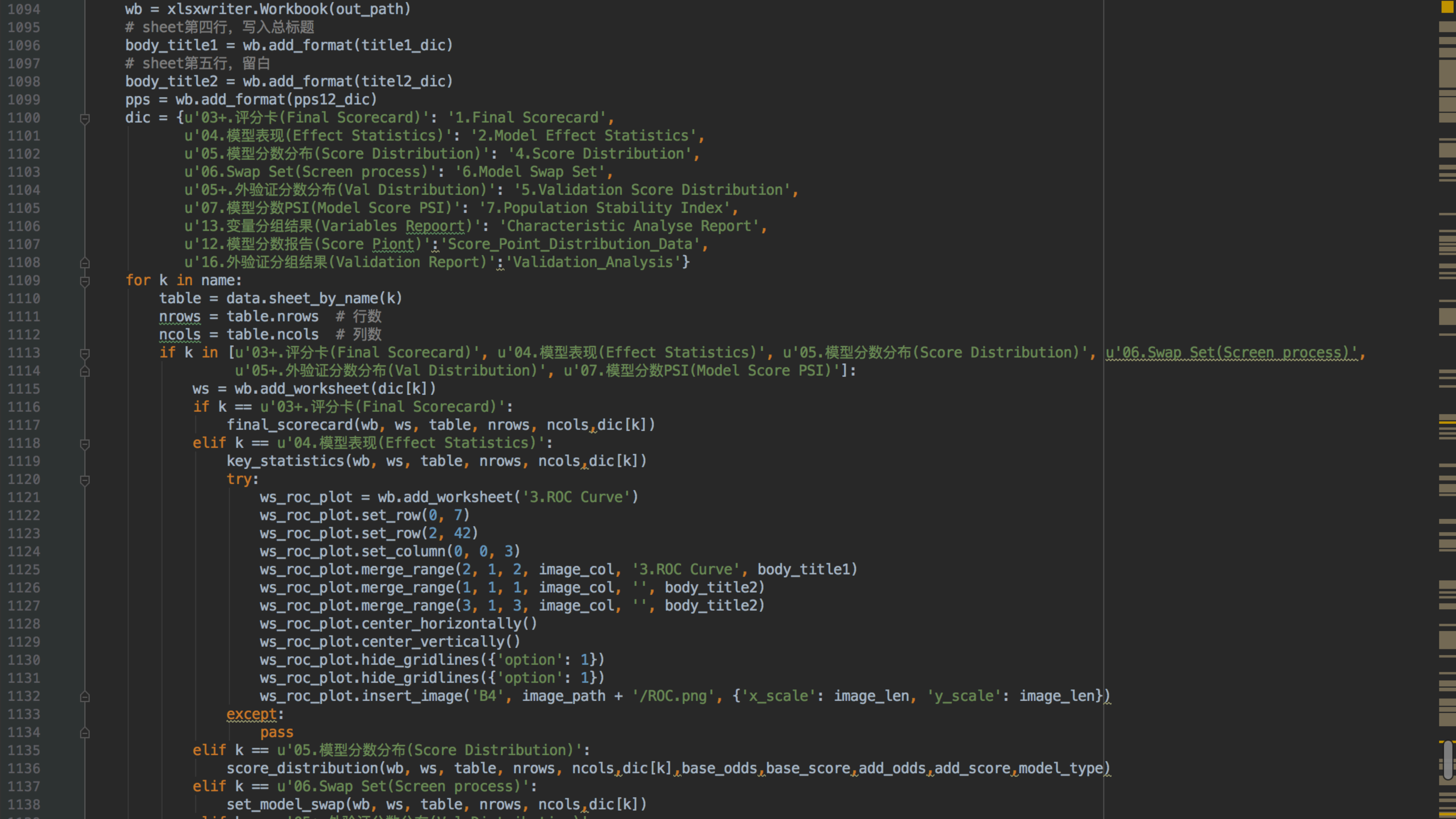


```
                              500, 2, 30, model_type)
        # save data
        try:
            train_data_woe.to_pickle(sample_path + '/%s_Data_WOE.pkl' % train_name)
            test_data_woe.to_pickle(sample_path + '/%s_Data_WOE.pkl' % test_name)
            oot_data_woe.to_pickle(sample_path + '/%s_Data_WOE.pkl' % oot_name)
            result.save(model_path + '/Model_Result.model')
        except:
            pass
        train_data_original.to_pickle(sample_path + '/%s_Data_Original.pkl' % train_name)
        test_data_original.to_pickle(sample_path + '/%s_Data_Original.pkl' % test_name)
        train_data_bin.to_pickle(sample_path + '/Characteristic_Bin_Data.pkl')
        score_bin.to_pickle(sample_path + '/Score_Bin_Data.pkl')
        if len(oot_data_original) > 0:
            oot_data_original.to_pickle(sample_path + '/%s_Data_Original.pkl' % oot_name)
```

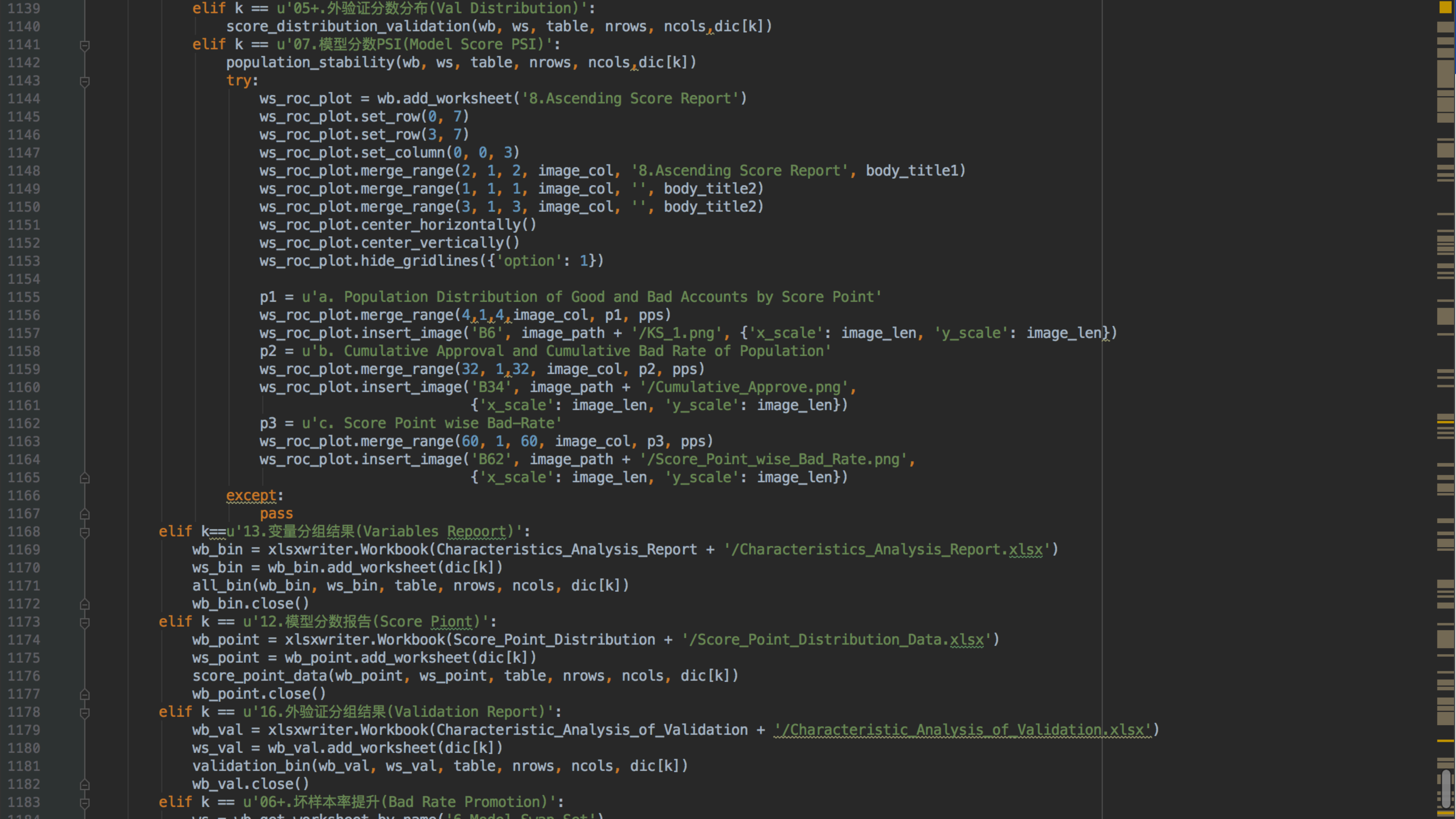
③模型结果美化输出



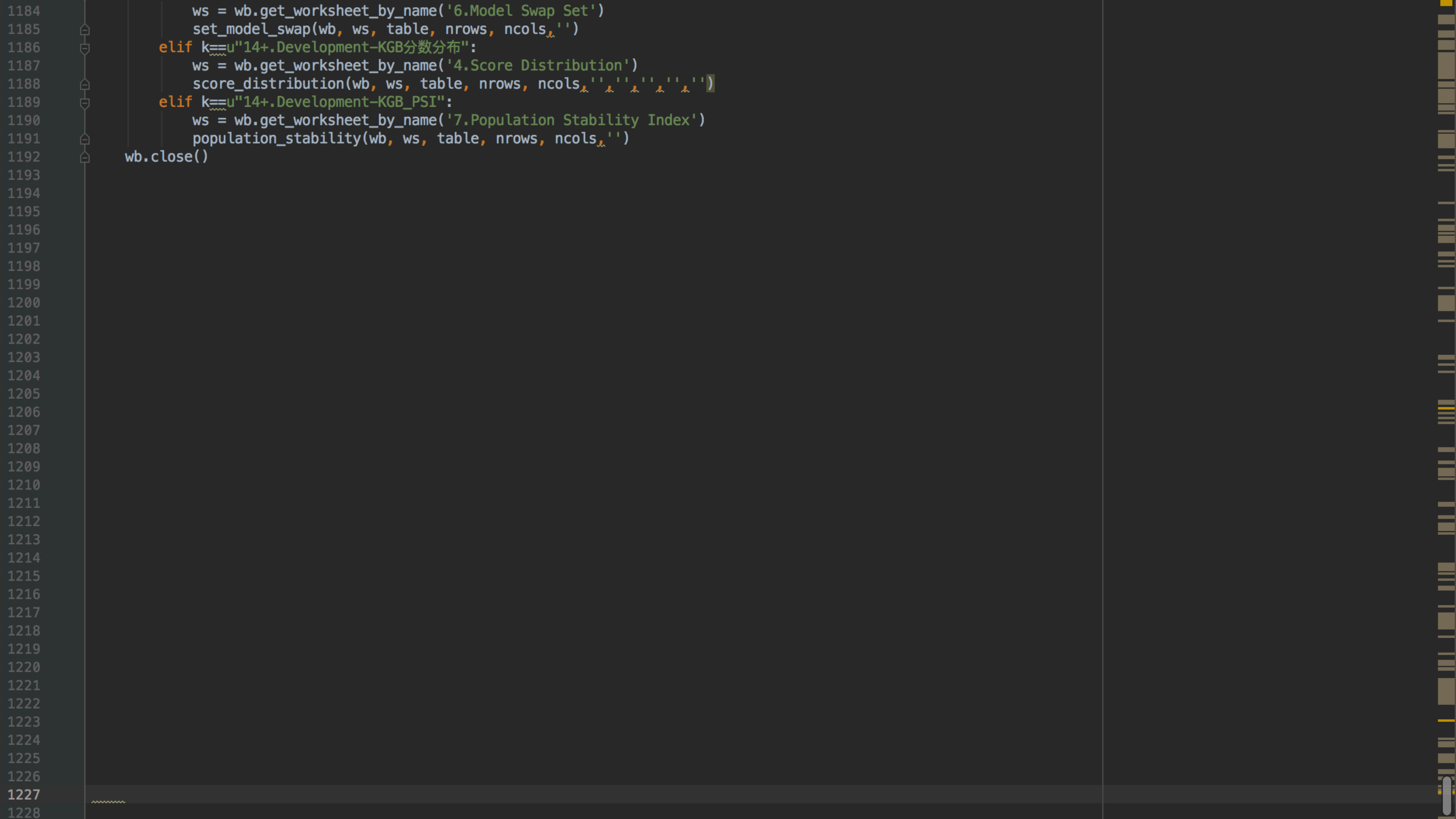
```
def model_result_beautify(image_path, in_path, out_path,Characteristics_Analysis_Report,
                          Characteristic_Analysis_of_Validation,Score_Point_Distribution,base_odds,base_score,add_odds,add_score,model_type):
    """
    Explanation
    ----------
    将模型汇总结果美化，可视化展示

    Parameters
    ----------
    image_path:string
        图片路径

    in_path:string
        模型汇总结果路径

    out_path:string
        美化后模型汇总结果存储路径

    Characteristics_Analysis_Report:string
        变量分析报告保存路径

    Characteristic_Analysis_of_Validation: string
        外验证集变量分析报告储存路径

    Score_Point_Distribution: string
        模型得分详细分析保存路径

    bad_odds:float, default=50
        基准好坏比

    base_score:int, default=500
        基准分

    add_odds:intfloat, default=2
        odds变化步长

    add_score:int, default=20
        odds变化一个步长，相应增长的分数
    """
    data = xlrd.open_workbook(in_path)
    name = data.sheet_names()
    wb = xlsxwriter.Workbook(out_path)
```



```
    wb = xlsxwriter.Workbook(out_path)
    # sheet第四行，写入总标题
    body_title1 = wb.add_format(title1_dic)
    # sheet第五行，留白
    body_title2 = wb.add_format(titel2_dic)
    pps = wb.add_format(pps12_dic)
    dic = {u'03+.评分卡(Final Scorecard)': '1.Final Scorecard',
           u'04.模型表现(Effect Statistics)': '2.Model Effect Statistics',
           u'05.模型分数分布(Score Distribution)': '4.Score Distribution',
           u'06.Swap Set(Screen process)': '6.Model Swap Set',
           u'05+.外验证分数分布(Val Distribution)': '5.Validation Score Distribution',
           u'07.模型分数PSI(Model Score PSI)': '7.Population Stability Index',
           u'13.变量分组结果(Variables Repoort)': 'Characteristic Analyse Report',
           u'12.模型分数报告(Score Piont)':'Score_Point_Distribution_Data',
           u'16.外验证分组结果(Validation Report)':'Validation_Analysis'}
    for k in name:
        table = data.sheet_by_name(k)
        nrows = table.nrows  # 行数
        ncols = table.ncols  # 列数
        if k in [u'03+.评分卡(Final Scorecard)', u'04.模型表现(Effect Statistics)', u'05.模型分数分布(Score Distribution)', u'06.Swap Set(Screen process)',
                 u'05+.外验证分数分布(Val Distribution)', u'07.模型分数PSI(Model Score PSI)']:
            ws = wb.add_worksheet(dic[k])
            if k == u'03+.评分卡(Final Scorecard)':
                final_scorecard(wb, ws, table, nrows, ncols,dic[k])
            elif k == u'04.模型表现(Effect Statistics)':
                key_statistics(wb, ws, table, nrows, ncols,dic[k])
                try:
                    ws_roc_plot = wb.add_worksheet('3.ROC Curve')
                    ws_roc_plot.set_row(0, 7)
                    ws_roc_plot.set_row(2, 42)
                    ws_roc_plot.set_column(0, 0, 3)
                    ws_roc_plot.merge_range(2, 1, 2, image_col, '3.ROC Curve', body_title1)
                    ws_roc_plot.merge_range(1, 1, 1, image_col, '', body_title2)
                    ws_roc_plot.merge_range(3, 1, 3, image_col, '', body_title2)
                    ws_roc_plot.center_horizontally()
                    ws_roc_plot.center_vertically()
                    ws_roc_plot.hide_gridlines({'option': 1})
                    ws_roc_plot.hide_gridlines({'option': 1})
                    ws_roc_plot.insert_image('B4', image_path + '/ROC.png', {'x_scale': image_len, 'y_scale': image_len})
                except:
                    pass
            elif k == u'05.模型分数分布(Score Distribution)':
                score_distribution(wb, ws, table, nrows, ncols,dic[k],base_odds,base_score,add_odds,add_score,model_type)
            elif k == u'06.Swap Set(Screen process)':
                set_model_swap(wb, ws, table, nrows, ncols,dic[k])
            elif k == u'05+.外验证分数分布(Val Distribution)':
```



```
            elif k == u'05+.外验证分数分布(Val Distribution)':
                score_distribution_validation(wb, ws, table, nrows, ncols,dic[k])
            elif k == u'07.模型分数PSI(Model Score PSI)':
                population_stability(wb, ws, table, nrows, ncols,dic[k])
                try:
                    ws_roc_plot = wb.add_worksheet('8.Ascending Score Report')
                    ws_roc_plot.set_row(0, 7)
                    ws_roc_plot.set_row(3, 7)
                    ws_roc_plot.set_column(0, 0, 3)
                    ws_roc_plot.merge_range(2, 1, 2, image_col, '8.Ascending Score Report', body_title1)
                    ws_roc_plot.merge_range(1, 1, 1, image_col, '', body_title2)
                    ws_roc_plot.merge_range(3, 1, 3, image_col, '', body_title2)
                    ws_roc_plot.center_horizontally()
                    ws_roc_plot.center_vertically()
                    ws_roc_plot.hide_gridlines({'option': 1})

                    p1 = u'a. Population Distribution of Good and Bad Accounts by Score Point'
                    ws_roc_plot.merge_range(4,1,4,image_col, p1, pps)
                    ws_roc_plot.insert_image('B6', image_path + '/KS_1.png', {'x_scale': image_len, 'y_scale': image_len})
                    p2 = u'b. Cumulative Approval and Cumulative Bad Rate of Population'
                    ws_roc_plot.merge_range(32, 1,32, image_col, p2, pps)
                    ws_roc_plot.insert_image('B34', image_path + '/Cumulative_Approve.png',
                                             {'x_scale': image_len, 'y_scale': image_len})
                    p3 = u'c. Score Point wise Bad-Rate'
                    ws_roc_plot.merge_range(60, 1, 60, image_col, p3, pps)
                    ws_roc_plot.insert_image('B62', image_path + '/Score_Point_wise_Bad_Rate.png',
                                             {'x_scale': image_len, 'y_scale': image_len})
                except:
                    pass
        elif k==u'13.变量分组结果(Variables Repoort)':
            wb_bin = xlsxwriter.Workbook(Characteristics_Analysis_Report + '/Characteristics_Analysis_Report.xlsx')
            ws_bin = wb_bin.add_worksheet(dic[k])
            all_bin(wb_bin, ws_bin, table, nrows, ncols, dic[k])
            wb_bin.close()
        elif k == u'12.模型分数报告(Score Piont)':
            wb_point = xlsxwriter.Workbook(Score_Point_Distribution + '/Score_Point_Distribution_Data.xlsx')
            ws_point = wb_point.add_worksheet(dic[k])
            score_point_data(wb_point, ws_point, table, nrows, ncols, dic[k])
            wb_point.close()
        elif k == u'16.外验证分组结果(Validation Report)':
            wb_val = xlsxwriter.Workbook(Characteristic_Analysis_of_Validation + '/Characteristic_Analysis_of_Validation.xlsx')
            ws_val = wb_val.add_worksheet(dic[k])
            validation_bin(wb_val, ws_val, table, nrows, ncols, dic[k])
            wb_val.close()
        elif k == u'06+.坏样本率提升(Bad Rate Promotion)':
            ws = wb.get_worksheet_by_name('6.Model Swap Set')
```



```
            ws = wb.get_worksheet_by_name('6.Model Swap Set')
            set_model_swap(wb, ws, table, nrows, ncols,'')
        elif k==u"14+.Development-KGB分数分布":
            ws = wb.get_worksheet_by_name('4.Score Distribution')
            score_distribution(wb, ws, table, nrows, ncols,'','','','','')
        elif k==u"14+.Development-KGB_PSI":
            ws = wb.get_worksheet_by_name('7.Population Stability Index')
            population_stability(wb, ws, table, nrows, ncols,'')
    wb.close()
```

评分卡全流程示例

同盾科技
www.tongdun.cn

THANKS