QUESTION 5.a

NOTEBOOK - Q6_GB_With_GridSearchCV_CS21MDS14025_final.ipynb

1. ***MISSING VALUE HANDLING :***

   1.1 Dropped the columns with more than 60% missing values

   remove any columns with more than 60% null values

```
]: not_null_cols = loan_train.columns[loan_train.isnull().mean() < 0.6]
   print('no of columns remaining after removing any columns with more than 60% null values : ',len(not_null_cols))#
   loan_train= loan_train.filter(not_null_cols,axis = 1)
   loan_train.isna().sum()
```

   1.2 For highly imbalanced classes, replaced with most frequent value:

```
In [272]: loan_train.pub_rec_bankruptcies.value_counts(dropna = False)

Out[272]: 0.0    22886
          1.0      994
          NaN      417
          2.0        4
          Name: pub_rec_bankruptcies, dtype: int64

In [273]: loan_train.pub_rec_bankruptcies = loan_train.pub_rec_bankruptcies.fillna(0)
          loan_train.pub_rec_bankruptcies.value_counts(dropna = False)

Out[273]: 0.0    23303
          1.0      994
          2.0        4
          Name: pub_rec_bankruptcies, dtype: int64
```

   1.3 For the remaining columns with very few missing values, did fillna with median:

```
loan_train = loan_train.fillna(loan_train.median())
loan_test = loan_test.fillna(loan_test.median())
```

2. ***FEATURE SELECTION :***

   2.1 Dropped the columns with more than 60% missing values as described above in step 1.1

   2.2 Removed the text columns that are rarely repeated . In other words, categorical columns with high cardinality.

```
: loan_train[str_cols].nunique()# checking the no of unique values for each object type column

: term                      2
  int_rate                361
  grade                     7
  sub_grade                35
  emp_title             19413
  emp_length               11
  home_ownership            4
  verification_status       3
  issue_d                  55
  pymnt_plan                1
  url                   24301
  desc                  16213
  purpose                  14
  title                 12698
  zip_code                783
  addr_state               50
  earliest_cr_line        495
  revol_util             1058
  initial_list_status       1
  last_pymnt_d            101
  last_credit_pull_d      102
  application_type          1
  dtype: int64
```

   Drop Non Categorical Descriptive Text/ Text Features with High Cardinality

```
: remove_cols = ['emp_title','url','desc','title','zip_code','addr_state'] ### THE COLUMNS that c
  remove_from_cat_features = remove_cols.copy()
```

2.3 Removed date columns with inconsistent date formats:
e.g. earliest_cr_line

earliest_cr_line
1-Feb
Feb-97
Mar-00
4-Jun
Aug-93

2.4 Created new features (Feature Engineering) & then removed the original features:

Calculate new feature from last_pymnt_d & last_credit_pull_d

```
loan_train['last_pymnt_d'] = pd.to_datetime(loan_train['last_pymnt_d'], format = '%d-%b') # 11- Jul format
loan_train['last_credit_pull_d'] = pd.to_datetime(loan_train['last_credit_pull_d'], format = '%d-%b')
loan_train['credit_pull_minus_pymnt'] = (loan_train['last_credit_pull_d'] - loan_train['last_pymnt_d']).dt.days
remove_cols.append('last_pymnt_d')
remove_cols.append('last_credit_pull_d')
remove_from_cat_features = remove_cols.copy()
```

All these previous feature selection steps reduced the no.of features from 111 to 45 :

Drop all columns in the list remove_cols

```
print(loan_train.shape)
loan_train.drop(remove_cols, axis =1, inplace =True)
print(loan_train.shape)
```

```
(24301, 55)
(24301, 45)
```

3. ***TRANSFORM CATEGORICAL COLUMNS into One-hot -encoded Binary columns***:
As mentioned in previous step 2.2, the categorical features with high cardinality were removed.
Also some of the Strings that were actually should have been numerical – were converted to numerical (as described in the STEP 4. "EXTRA PREPROCESSING STEPS to improve the performance") The remaining categorical features were converted to one -hot -encoded binary features:

```
final_cols_b4_enc = loan_train.columns
```

```
print(loan_train.shape)
dummy_cols = pd.get_dummies(loan_train, prefix=None, prefix_sep='_', dummy_na=False, columns= str_cols)
loan_train = pd.concat([loan_train.drop(str_cols,axis=1, inplace = True), dummy_cols],axis=1)
print(loan_train.shape)
```

```
(24301, 45)
(24301, 103)
```

4. ***EXTRA PREPROCESSING STEPS to improve the performance :***
4.1 : Feature Engineering – as described in the step 2.4
4.2 : Convert the columns into more useful format : e.g. – from "65 Years" string to 65 numerical value:

```
loan_train['emp_length'].value_counts()# before pre processing
```

```
10+ years    5315
< 1 year     2816
2 years      2806
3 years      2675
4 years      2223
5 years      2087
1 year       2081
6 years      1385
7 years      1108
8 years       942
9 years       823
Name: emp_length, dtype: int64
```

```
#loan_train['emp_length'] = [pd.to_numeric(x.strip('<').lstrip().split('+')[0].split()[0], errors = 'ignore') for x in loan_trai
loan_train['emp_length'] = [pd.to_numeric(x.split('+')[0].split()[0], errors = 'ignore') for x in loan_train['emp_length'].astyp
#loan_train['emp_length'].value_counts()
loan_train['emp_length'].replace('<',0, inplace = True)
remove_from_cat_features.append('emp_length')
loan_train['emp_length'].value_counts()
```

```
10    5315
0     2816
2     2806
3     2675
4     2223
5     2087
1     2081
6     1385
7     1108
8      942
```

Similarly a string like “10%” was converted into a numerical value of 10:

```
loan_train['int_rate'] = [float(x.split('%')[0]) for x in loan_train['int_rate'].astype(str) ]#reve
loan_train['revol_util'] = [float(x.split('%')[0]) for x in loan_train['revol_util'].astype(str) ]
remove_from_cat_features.append('int_rate')
remove_from_cat_features.append('revol_util')
```

QUESTION 5.b

Precision & Recall of each model (i.e hyperparameter combination)

Since I have tried ~ 170 hyperparameter combination , please check the IPython notebook & do a Ctrl+F to search for this header : “Hyperparameter Selection via Grid Search CV”

NOTE – I selected n_jobs = -1 to utilize all cores in my PC. The actual execution time will depend on your machine.

Hyperparameter Selection via Grid Search CV

```
start = time.time()
precision_scorer = make_scorer(precision_score, zero_division=0)

custom_scoring = {"accuracy": "accuracy", "precision": precision_scorer, "recall": "recall", "f1": "f1"}


parameter_list = {'n_estimators':[50,100,150,200], 'max_depth':[3,4,5], 'max_features':['sqrt','log2'],
                  'learning_rate':[0.4, 0.5,0.6, 0.7, 0.8,1,1.2]} # the set of hyper parameters checked
models = GridSearchCV(GradientBoostingClassifier(), parameter_list,scoring=custom_scoring , refit="accuracy", n_jobs =-1)
# n_jobs =-1 uses all the hardware processors. Depending on your machine, might take more time
models.fit(X_train, loan_train['target'])
end = time.time()
print('time in minutes', (end - start)/60)
```

```
time in minutes 5.829510529836019
```

The following cell shows the hyperparameters used for each model :

```
In [296]: models.cv_results_['params']

Out[296]: [{'learning_rate': 0.4,
            'max_depth': 3,
            'max_features': 'sqrt',
            'n_estimators': 50},
           {'learning_rate': 0.4,
            'max_depth': 3,
            'max_features': 'sqrt',
            'n_estimators': 100},
           {'learning_rate': 0.4,
            'max_depth': 3,
            'max_features': 'sqrt',
            'n_estimators': 150},
           {'learning_rate': 0.4,
            'max_depth': 3,
            'max_features': 'sqrt',
            'n_estimators': 200},
           {'learning_rate': 0.4,
            'max_depth': 3,
            'max_features': 'log2',
```

The precision, recall, & accuracy of each model is displayed in this format in the notebook :

```
In [295]: models.cv_results_['mean_test_accuracy']

Out[295]: array([0.9927163 , 0.99456814, 0.99539114, 0.99604956, 0.98971223,
       0.99296319, 0.9945681 , 0.99321038, 0.99399202, 0.99510308,
       0.99518536, 0.9959261 , 0.99156415, 0.99465048, 0.99456812,
       0.99493845, 0.99444465, 0.99469156, 0.9960907 , 0.99596726,
       0.99238714, 0.99378623, 0.99423893, 0.99514421, 0.99395081,
       0.99477391, 0.99547345, 0.99090553, 0.9909057 , 0.99407425,
       0.99473271, 0.99522653, 0.993704  , 0.99234622, 0.98428001,
       0.99613187, 0.99267513, 0.99102917, 0.98695493, 0.99526766,
       0.99386853, 0.98440347, 0.99580266, 0.9958438 , 0.99263404,
       0.99366279, 0.9945681 , 0.99506192, 0.99316895, 0.99304588,
       0.99098782, 0.99604955, 0.99205783, 0.99304562, 0.99510305,
       0.92104327, 0.99181095, 0.98773685, 0.99584381, 0.99543231,
       0.99283981, 0.99263398, 0.93909485, 0.8764611 , 0.99370397,
       0.99049401, 0.99580265, 0.99559696, 0.99119375, 0.98658457,
       0.99473274, 0.96790141, 0.99399202, 0.98403355, 0.97888904,
       0.92198582, 0.9923048 , 0.99386858, 0.99012366, 0.99485615,
       0.99288086, 0.98864299, 0.88938291, 0.93996431, 0.99041216,
       0.98432119, 0.97851936, 0.9951031 , 0.99362163, 0.92255169,
       0.89156401, 0.97296767, 0.98827212, 0.96494036, 0.96115249,
       0.95596729, 0.99111133, 0.9846504 , 0.94847815, 0.9635422 ,
       0.99279862, 0.97720192, 0.99456811, 0.92884894, 0.9904118 ,
       0.9904529 , 0.90519044, 0.86843639, 0.99197563, 0.97267744,
       0.94559693, 0.88753109, 0.9876549 , 0.95494052, 0.96061744,
       0.75309003, 0.98382799, 0.86585493, 0.77704356, 0.88893416,
       0.99176993, 0.97424065, 0.65840625, 0.8238369 , 0.99094683,
       0.97308711, 0.93510992, 0.73519489, 0.93983587, 0.86123591,
       0.82700536, 0.65726506, 0.98502112, 0.91370536, 0.68525113,
       0.74548412, 0.96967108, 0.84810846, 0.60510655, 0.8286903 ,
       0.96559797, 0.86070336, 0.82610321, 0.80843908, 0.95913624,
       0.93728454, 0.79140427, 0.91732855, 0.98592643, 0.9521004 ,
       0.61064732, 0.708544  , 0.97802528, 0.6429422 , 0.81367546,
       0.84716905, 0.96477516, 0.75959889, 0.79865207, 0.700296  ,
       0.91172911, 0.77041538, 0.73104863, 0.82354422, 0.92205873,
       0.8179514 , 0.82552046, 0.83914126])
```

How the model performance changes with changing no of trees

The results are available under the below header : (“Printing the Precision , recall & accuracy for different tree numbers”)

SUMMARY OF THE PERFORMANCE EXPLORATION WITH CHANGING NO OF ESTIMATORS :

There is a significant drop in performance close to n_estimators =330.

But other than that , we see that for a lower no of trees, recall still remains high whereas Accuracy , Precision & F1 get reduced.

Printing the Precision , recall & accuracy for different tree numbers

Reported on the Validation Set(Not Test Set) - as Hyper param selection needs to be done

```
]: import gc
   gc.collect()
```

```
]: 3816
```

```
]: # GradientBoostingClassifier(learning_rate=0.5, max_depth=4, max_features='sqrt', n_estimators=200)
   import time
   start = time.time()
   parameter_list = {'n_estimators':[7,10,20,30,40,50,70,100,150,200, 220, 250, 270,300,330, 360,400], 'max_depth':[4], 'max_feature
                     'learning_rate':[0.5]}
   model_tree = GridSearchCV(GradientBoostingClassifier(), parameter_list,scoring=custom_scoring , refit="accuracy", n_jobs =-1)
   model_tree.fit(X_train, loan_train['target'])
   end = time.time()
   # n_jobs =-1 uses all the hardware processors. Depending on your machine, might take more time
   print('time in minutes', (end - start)/60)
```

```
time in minutes 1.0235695083936056
```

```
In [299]: for item in model_tree.cv_results_['params']:
              print(item['n_estimators'])
```

```
7
10
20
30
40
50
70
100
150
200
220
250
270
300
330
360
400
```

```
In [300]: model_tree.cv_results_['mean_test_precision']
```

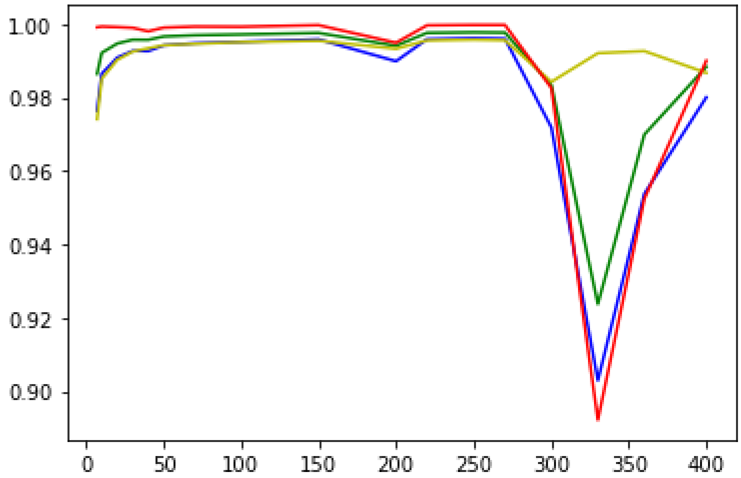
```
Out[300]: array([0.9740947 , 0.9851105 , 0.99024753, 0.99251399, 0.99340746,
                 0.99422223, 0.99460298, 0.99507667, 0.99546093, 0.99326939,
                 0.99555501, 0.99574577, 0.99560231, 0.98427057, 0.99207772,
                 0.99264433, 0.98671639])
```

```
In [301]: model_tree.cv_results_['mean_test_recall']
```

```
Out[301]: array([0.99913572, 0.99932781, 0.99918377, 0.99899168, 0.99803143,
                 0.99903966, 0.99932779, 0.9992798 , 0.99971191, 0.99490998,
                 0.99966392, 0.99975993, 0.99975992, 0.98256906, 0.8921747 ,
                 0.95241302, 0.99001203])
```

```
In [302]: model_tree.cv_results_['mean_test_f1']
```

```
Out[302]: array([0.98643788, 0.99216286, 0.99469501, 0.99574124, 0.99571295,
                 0.99662391, 0.99695902, 0.99717353, 0.99758119, 0.99407826,
                 0.99760466, 0.99774832, 0.99767649, 0.98337772, 0.92374708,
                 0.96991335, 0.98836008])
```

In [303]:
```
model_tree.cv_results_['mean_test_accuracy']
```
Out[303]:
```
array([0.97642054, 0.98646129, 0.99086458, 0.99267515, 0.99263396,
       0.99419773, 0.9947739 , 0.9951442 , 0.99584379, 0.9898767 ,
       0.99588496, 0.99613185, 0.99600839, 0.971852  , 0.90294118,
       0.95374507, 0.98004132])
```
In [304]:
```
model_tree.best_estimator_
```
Out[304]:
```
GradientBoostingClassifier(learning_rate=0.5, max_depth=4, max_features='
                           n_estimators=250)
```
In [305]:
```
x = [7,10,20,30,40,50,70,100,150,200, 220, 250, 270,300,330, 360,400]
plt.plot(x, model_tree.cv_results_['mean_test_accuracy'],'b')
plt.plot(x, model_tree.cv_results_['mean_test_f1'],'g')
plt.plot(x, model_tree.cv_results_['mean_test_precision'],'y')
plt.plot(x, model_tree.cv_results_['mean_test_recall'],'r')
plt.show()
```


BEST Performance vs Decision Tree :

Please search for the Header : “Comparison with Decision Tree”

We can see the optimally selected GB model gives us :

```
Test Accuracy 0.9960072849537686
Test Precision 1.0
Test Recall 0.9953201970443349
```
while a Decision Tree gives us :

```
Test Accuracy 0.991874474642757
Test Precision 0.9948032665181886
Test Recall 0.9956245356228846
```

0 50 100 150 200 250 300 350 400

In [306]:
```
selected_model = GradientBoostingClassifier(learning_rate=0.5, max_depth=4, max_features='sqrt',n_estimators=360)
selected_model.fit(X_train,loan_train['target'])
y_predicted = selected_model.predict(X_test)
print('Test Accuracy',accuracy_score(y_predicted, loan_test['target']))
print('Test Precision',precision_score(y_predicted, loan_test['target']))
print('Test Recall',recall_score(y_predicted, loan_test['target']))
```

```
Test Accuracy 0.9960072849537686
Test Precision 1.0
Test Recall 0.9953201970443349
```

Comparison with Decision Tree

In [307]:
```
DT = DecisionTreeClassifier()
DT = DT.fit(X_train,loan_train['target'])
y_predicted = DT.predict(X_test)
print('Test Accuracy',accuracy_score(y_predicted, loan_test['target']))
print('Test Precision',precision_score(y_predicted, loan_test['target']))
print('Test Recall',recall_score(y_predicted, loan_test['target']))
```

```
Test Accuracy 0.991874474642757
Test Precision 0.9948032665181886
Test Recall 0.9956245356228846
```

In [ ]: