Feature Selection and Analysis:

Initial Dataset contained 115 Variables and 188181 rows. There were many obvious removals present in the dataset like zip code, State and there were other variables where there was the monotonous distribution of values, there were some variables with around 90% of missing values. These variables are better off being removed rather than trying to impute the missing records.

Following is the summary of Initial removal of Variables:

- Deleted 'desc' which stands for description or purpose for the loan amount for the customer: Description is a text field, so decided to remove
- deleted 'title'- too many categories in this variable. Interpretation of Model would be difficult
- deleted 'zipcode': Just a geographical information variable.
- deleted 'annual_inc_joint', dti_joint', 'verification_status_joint': No value found
- deletd 'open_acc_6m' , open_il_6m, open_il_12m , open_il_24m, mths_since_rcnt_il, total_bal_il , till the variables 'all_util' : These variables appear from “BL” to “BV” in the original dataset and have no values present in their fields
- deleted 'inq_fi', 'total_cu' and 'inq_last' for the same reason as above
- deleted pymnt-plan since every element is "n" and will not add any meaning to the model. there is no variation in data
- deleted 'mnths_since_last_record' since 170707 out of 188181 records are have missing values and imputing would be harmful to the model
- deleted out_prncp since data is skewed with 171456 out of 188181 records having value 0
- deleted out_prncp_inv with the same above reason

- deleted total_rec_late_fee since 182832 out of 188181 records are of value '0'.

- similar reason: recoveries

- similar reason: collection_recovery fee

After Due thought process over which variable must be chosen as the target variable, I settled with Loan_status. Though I did some modifications to the original distribution of the data.

I first categorized 'loan_status' variable into good or bad loans:

Good loan: value 1(Fully paid, current)

Bad loan: value 0 (all others like default, late, charged off etc.).

Even after deleting above 30 variables, I was still left with more than 70 variables seemingly containing too many correlated variables. This is the most time-consuming process for me.

I first tried applying Boruta algorithm, which applies series of random forest techniques within it and spits out three categories of results:

Confirmed

Tentative

Rejected

But as the data consisted correlated variables, it was still confirming 70 odd variables. So, I had to resort to Dimension Reduction "PCA".

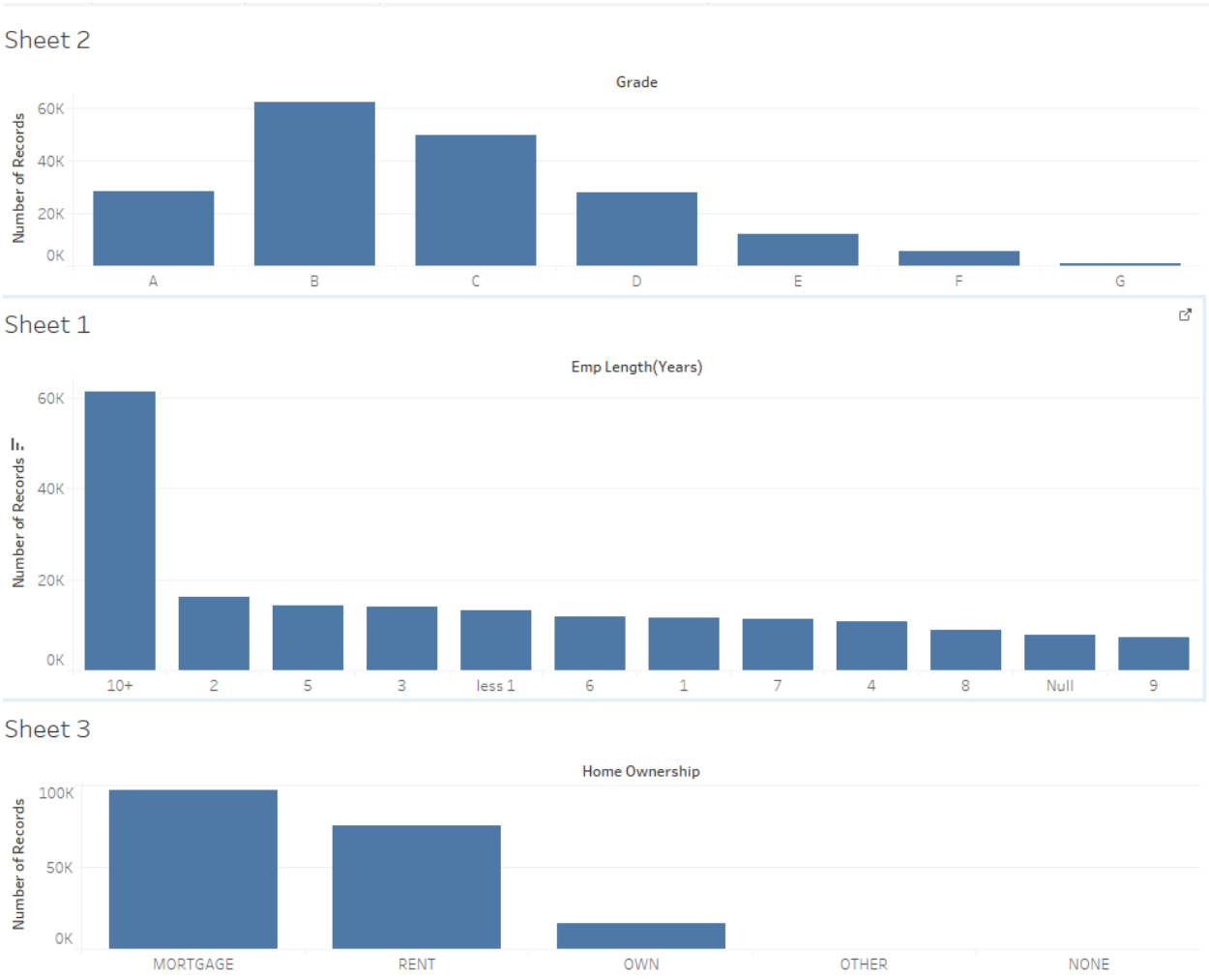
But there were categorical variables in the data. So, I separated the numerical variables into a separate csv file and kept the categorical variables separately.

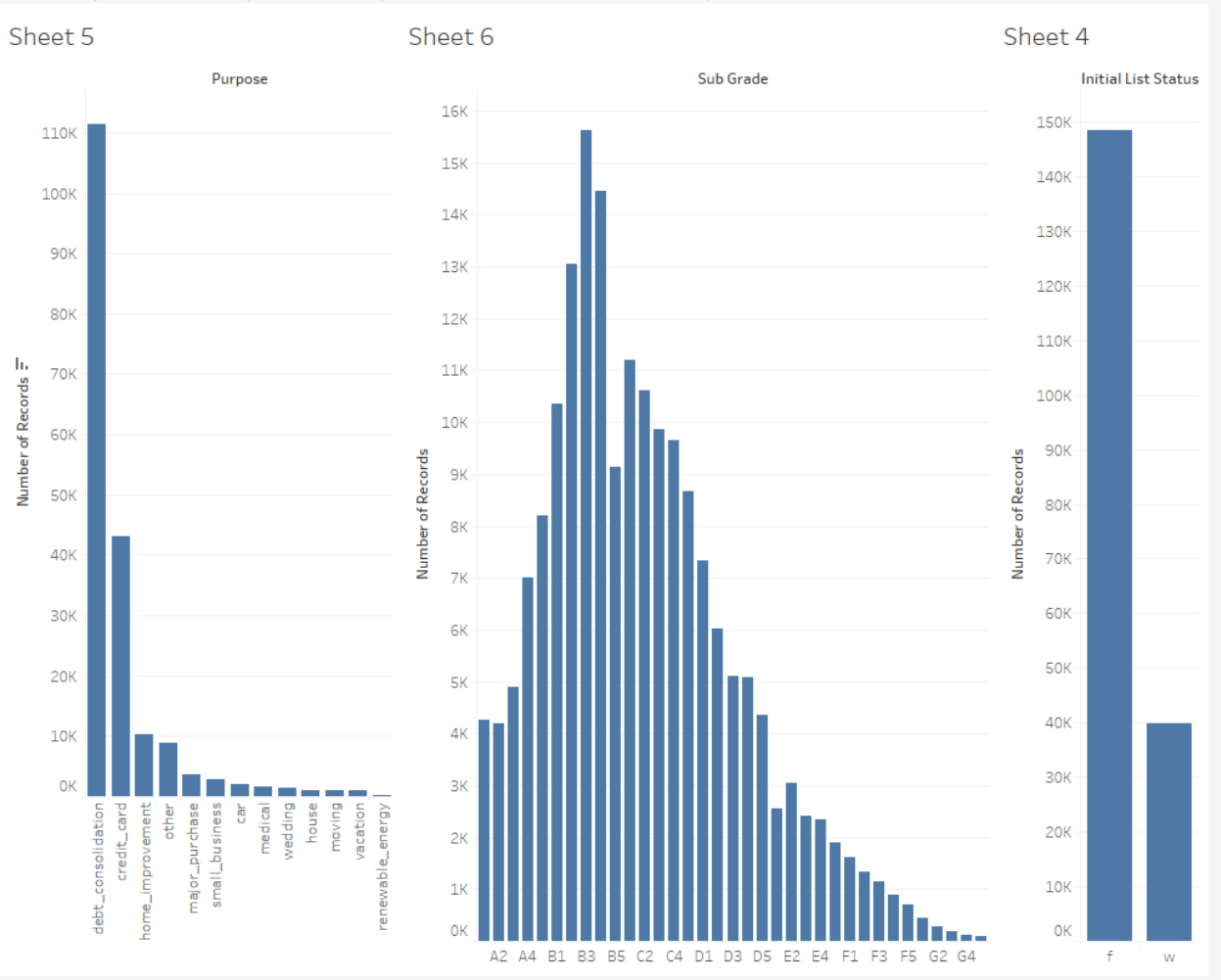
I applied PCA on numerical variables and as expected there was huge correlation and redundancy present in the data. The first 7 PCA's could explain 99.5% of the variance in the data. So, I removed the rest of the PCA's.

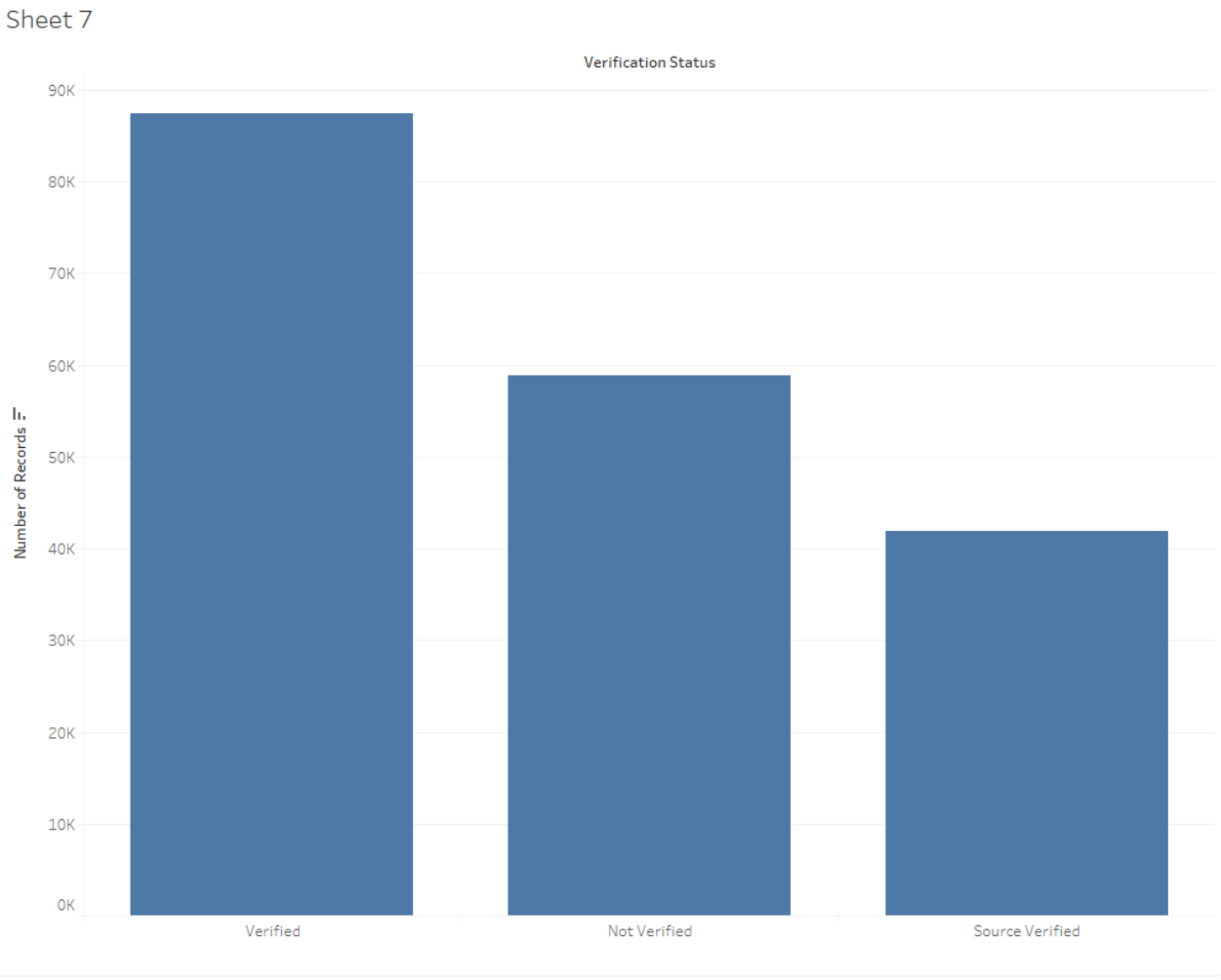
Since PCA is a technique where a PCA is the linear combination of the other variables, we can safely join the newly formed PCA data with the separated categorical variable dataset. So, exported the CSV file from python and joined it with the categorical dataset to result in 14 features and 1 target Variable.

All the PCA s from 1 to 7 are relevant, so didn't do distribution analysis on them

The categorical features distribution:




Sheet 5
Purpose
Number of Records
110K
100K
90K
80K
70K
60K
50K
40K
30K
20K
10K
0K
debt_consolidation
credit_card
home_improvement
other
major_purchase
small_business
car
medical
wedding
house
moving
vacation
renewable_energy
Sheet 6
Sub Grade
Number of Records
16K
15K
14K
13K
12K
11K
10K
9K
8K
7K
6K
5K
4K
3K
2K
1K
0K
A2 A4 B1 B3 B5 C2 C4 D1 D3 D5 E2 E4 F1 F3 F5 G2 G4
Sheet 4
Initial List Status
Number of Records
150K
140K
130K
120K
110K
100K
90K
80K
70K
60K
50K
40K
30K
20K
10K
0K
f
w



Exercise:

Since the data is ready for further modelling, I initially split the data into two parts: 75% and 25%.

I then split the original 75% into 80-20 and performed 10-fold cross validation on 80 split date. Trained all the required models on the 80. I then tested the trained models on 20% data of original 75% and finally tested on original hold out data of 25%.

For Cross validation:

Among the 4 classifiers, Decision tree had the least accuracy of about 77% and the Logistic regression, Naïve Bayes, K-Nearest Neighbor had accuracies more or less the same ranging from 82.5 % KNN to 84.5% Logistic regression.

3. PCA is a process where the Component is a linear combination of other variables and all those variables would be un-correlated, so didn't perform any further combination of multiple variables.

4. After training the models on the training set, I tested them on 0.2 split of original 75% data. Following are the results:

Logistic Regression:

```
# Testing the model on 20% data of Initial 75% Data

test_prediction_regr = regr.predict(X_test1)
print ('Accuracy1_regr:', accuracy_score(Y_test1, test_prediction_regr))
print ('F1 score_regr:', f1_score(Y_test1, test_prediction_regr,average='weighted'))
print ('Recall_regr:', recall_score(Y_test1, test_prediction_regr,
                                    average='weighted'))
print ('Precision_regr:', precision_score(Y_test1, test_prediction_regr,
                                          average='weighted'))
```

```
Accuracy1_regr: 0.847096751337
F1 score_regr: 0.776973816457
Recall_regr: 0.847096751337
Precision_regr: 0.717572906126
```

Naïve Bayes:

```
# Testing the model on 20% data of Initial 75% Data

test_prediction_bayes = bayes_clf.predict(X_test1)
print ('Accuracy1_bayes:', accuracy_score(Y_test1, test_prediction_bayes))
print ('F1 score_bayes:', f1_score(Y_test1, test_prediction_bayes,average='weighted'))
print ('Recall_bayes:', recall_score(Y_test1, test_prediction_bayes,
                                     average='weighted'))
print ('Precision_bayes:', precision_score(Y_test1, test_prediction_bayes,
                                           average='weighted'))
```

```
Accuracy1_bayes: 0.836752045913
F1 score_bayes: 0.79414346253
Recall_bayes: 0.836752045913
Precision_bayes: 0.782023285699
```

Decision Tree:

```
# Testing the model on 20% data of Initial 75% Data

test_prediction_Tree = infoGain_clf.predict(X_test1)
print ('Accuracy1_Tree:', accuracy_score(Y_test1, test_prediction_Tree))
print ('F1 score_Tree:', f1_score(Y_test1, test_prediction_Tree,average='weighted'))
print ('Recall_Tree:', recall_score(Y_test1, test_prediction_Tree,
                                    average='weighted'))
print ('Precision_Tree:', precision_score(Y_test1, test_prediction_Tree,
                                          average='weighted'))
```

```
Accuracy1_Tree: 0.765047649414
F1 score_Tree: 0.768313006718
Recall_Tree: 0.765047649414
Precision_Tree: 0.771731406816
```

K-Nearest Neighbor:

```
# Testing the model on 20% data of Initial 75% Data

test_prediction_KNN = knearest_clf.predict(X_test1)
print ('Accuracy1_KNN:', accuracy_score(Y_test1, test_prediction_KNN))
print ('F1 score_KNN:', f1_score(Y_test1, test_prediction_KNN,average='weighted'))
print ('Recall_KNN:', recall_score(Y_test1, test_prediction_KNN,
                                   average='weighted'))
print ('Precision_KNN:', precision_score(Y_test1, test_prediction_KNN,
                                         average='weighted'))
```

```
Accuracy1_KNN: 0.825521663655
F1 score_KNN: 0.783824257634
Recall_KNN: 0.825521663655
Precision_KNN: 0.762787071259
```

5. The same trained models were finally tested on the original hold-out data of 25% and the results seem to be normal:

Following are the results:

Logistic Regression:

```
# Testing the model on initial 25% data

test_prediction_regr = regr.predict(X_test)
print ('Accuracy1_regr:', accuracy_score(Y_test, test_prediction_regr))
print ('F1 score2_regr:', f1_score(Y_test, test_prediction_regr,average='weighted'))
print ('Recall2_regr:', recall_score(Y_test, test_prediction_regr,
                                     average='weighted'))
print ('Precision2_regr:', precision_score(Y_test, test_prediction_regr,
                                           average='weighted'))
```

```
Accuracy1_regr: 0.847234621434
F1 score2_regr: 0.777168742322
Recall2_regr: 0.847234621434
Precision2_regr: 0.717806503757
```

Naïve Bayes:

```
# Testing the model on initial 25% data
test_prediction_bayes = bayes_clf.predict(X_test)
print ('Accuracy1_bayes:', accuracy_score(Y_test, test_prediction_bayes))
print ('F1 score2_bayes:', f1_score(Y_test, test_prediction_bayes,average='weighted'))
print ('Recall2_bayes:', recall_score(Y_test, test_prediction_bayes,
                                      average='weighted'))
print ('Precision2_bayes:', precision_score(Y_test, test_prediction_bayes,
                                            average='weighted'))
```

```
Accuracy1_bayes: 0.837478212813
F1 score2_bayes: 0.794626932352
Recall2_bayes: 0.837478212813
Precision2_bayes: 0.783028589759
```

Decision Tree:

```
# Testing the model on initial 25% data
test_prediction_Tree = infoGain_clf.predict(X_test)
print ('Accuracy1_Tree:', accuracy_score(Y_test, test_prediction_Tree))
print ('F1 score2_Tree:', f1_score(Y_test, test_prediction_Tree,average='weighted'))
print ('Recall2_Tree:', recall_score(Y_test, test_prediction_Tree,
                                     average='weighted'))
print ('Precision2_Tree:', precision_score(Y_test, test_prediction_Tree,
                                           average='weighted'))
```

```
Accuracy1_Tree: 0.768949538749
F1 score2_Tree: 0.772043405184
Recall2_Tree: 0.768949538749
Precision2_Tree: 0.77528250945
```

K-Nearest Neighbor:

```
# Testing the model on initial 25% data

test_prediction_KNN = knearest_clf.predict(X_test)
print ('Accuracy1_KNN:', accuracy_score(Y_test, test_prediction_KNN))
print ('F1 score2_KNN:', f1_score(Y_test, test_prediction_KNN,average='weighted'))
print ('Recall2_KNN:', recall_score(Y_test, test_prediction_KNN,
                                    average='weighted'))
print ('Precision2_KNN:', precision_score(Y_test, test_prediction_KNN,
                                          average='weighted'))
```

```
Accuracy1_KNN: 0.827147897802
F1 score2_KNN: 0.785353737143
Recall2_KNN: 0.827147897802
Precision2_KNN: 0.76527171361
```

So, Logistic regression and Naïve Bayes classifier gave the top 2 accuracies. It depends on the analytics problem to decide which among the two models are to be chosen. If the purpose is prediction and interpretation of variables, then I would go for Naïve Bayes rather than Logistic Regression since interpretation using PCA variables would be a little problematic.