Feature Engineering ¶

Feature engineering is especially important for financial dataset, which is very noisy in nature.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures

import warnings
warnings.filterwarnings('ignore')

pd.set_option('display.width', 500)
pd.set_option('display.max_columns', 500)

% matplotlib inline
```

In [2]:
```
df_eda = pd.read_csv("data/output_eda.csv", parse_dates=['issue_d'])
df_clean = df_eda.copy(deep=True)
```

Reduce Sample Bias

As we have seen in the EDA - Good Loans vs Bad Loans part, most of loans in recent five years are all new loans with most of them as current status. However, as time goes by, some of the loans may become bad loans. In order to avoid this sample bias, we decide to drop loans with issue dates in recent five years.

In [3]:
```
df_clean = df_clean[df_clean['year'] < 2013]
```

Generate Response Variable

According the lending club website, we define the following loan status as **bad loans**:

- **Default**: Loan has not been current for an extended period of time.
- **Charged Off**: Loan for which there is no longer a reasonable expectation of further payments. Upon Charge Off, the remaining principal balance of the Note is deducted from the account balance.
- **In Grace Period**: Loan is past due but within the 15-day grace period.
- **Late (16-30)**: Loan has not been current for 16 to 30 days.
- **Late (31-120)**: Loan has not been current for 31 to 120 days.
- **Does not meet the credit policy. Status:Charged Off**

In [4]:
```
bad_loan = set(["Charged Off",
               "Default",
               "Does not meet the credit policy. Status:Charged Off",
               "In Grace Period",
               "Late (16-30 days)",
               "Late (31-120 days)"])

df_clean['response'] = df_clean['loan_status'].apply(lambda x: 1 if x in bad_
df_clean.drop(['loan_status'], axis=1, inplace=True)
```

Deal with Missing Value

We replaced missing value with mean value in each loan grade bucket.

In [5]:
```
# Identify the columns that are numeric without 100% coverage
col_count = df_clean.describe().loc['count',]
col_count_nrm = col_count / max(col_count)

# Replace these columns with mean value in each loan grade
col_missing = ['tot_cur_bal', 'tot_coll_amt', 'dti', 'inq_last_6mths']
for col in col_missing:
    df_clean[col] = df_clean.groupby("grade")[col].transform(lambda x: x.fill
```

Deal with Dates

Some features of types of dates could be helpful to predict the loan default probability, such as **earliest credit line date** for the applicant. Typically the longer the history of the applicant's credit line, the more confidence we have on his/her FICO score, and the lower probability of default in general. However, we need to anchor the earliest credit line date relative to the loan issue date, as that's the information we know at initial investment stage.

We also cleaned up other date columns that should not be treated as features.

In [6]:
```
df_clean['earliest_cr_line'] = pd.to_datetime(df_clean['earliest_cr_line'])
df_clean['earliest_cr_line'] = df_clean['issue_d'] - df_clean['earliest_cr_li
df_clean['earliest_cr_line'] = df_clean['earliest_cr_line'].apply(lambda x: x
df_clean.drop(['issue_d'], axis=1, inplace=True)
```

Deal with Categorical Variables

There are two types of categorical variables that we need to engineer.

- One type of categorical features contain ordinal order, and we want to maitain that order, like grade (In term of loan quality, A > B > ... > G), and employement length (10+ years > 9 years > ... > 1 year).
- The other type of cateorical features don't have any ordinal order, like purpose of the loan, application type, etc. We will use one-hot-encoding to create dummy variables for this type of categories.

In [7]:
```
# Drop States
df_clean.drop(['addr_state'], axis=1, inplace=True)
```

In [8]:
```
# Loan grade
grade_map = {"A": 1, "B": 2, "C": 3, "D": 4, "E": 5, "F": 6, "G": 7}
df_clean['grade'] = df_clean['grade'].map(grade_map)
```

In [9]:
```
# Employement Length
df_clean['emp_length'] = df_clean['emp_length'].apply(lambda x: '0 year' if x

# Extract numerical value
import re
df_clean['emp_length'] = df_clean['emp_length'].apply(lambda x: int(re.findal
                                                       if isinstance(x, str) els

# Replace missing employment length as mean value in each sub_grade bucket
df_clean['emp_length'] = df_clean.groupby("grade")['emp_length'].transform(la
```

In [10]:
```
# Application Type
application_type_map = {'Individual': 1, 'Joint App': 0}
df_clean['application_type'] = df_clean['application_type'].map(application_t
```

In [11]:
```
# Verification Status
df_clean.rename({'verification_status': 'ver'}, axis=1, inplace=True)

ver_map = {
    "Source Verified": "Source_Verified",
    "Not Verified": "Not_Verified",
    "Verified": "Verified"
}
df_clean['ver'] = df_clean['ver'].map(ver_map)
```

In [12]:
```
# One-hot-encoding
dummy_list = ['home_ownership', 'ver', 'purpose']
df_clean = pd.get_dummies(df_clean, columns=dummy_list, drop_first=True)
```

Add Polynomial Terms

There might be non-linear effects between the response variable and the predictors, and thus we added polynomial and interaction terms to some of the important non-binary features.

In [13]:
```
df_clean.reset_index(inplace=True, drop=True)

poly_vars = ['int_rate', 'annual_inc', 'revol_util', 'term', 'dti', 'loan_amn
df_poly = df_clean[poly_vars]

tra = PolynomialFeatures(degree=3, include_bias=False)
temp = tra.fit_transform(df_poly)
df_temp = pd.DataFrame(temp, columns=tra.get_feature_names(df_poly.columns))

df_clean.drop(poly_vars, axis=1, inplace=True)
df_clean = df_clean.merge(df_temp, left_index=True, right_index=True)
```

Generate Additional Features

We generated an addtional feature that we believe would be helpful.

- The range of FICO score (fico_rng): (High FICO - Low FICO) / Mean FICO. This feature is designed to capture the range of FICO score.

In [14]:
```
df_clean['fico_rng'] = (df_clean['fico_range_high'] - df_clean['fico_range_lo
df_clean.drop(['fico_range_low', 'fico_range_high'], axis=1, inplace=True)
```

In [15]:
```
assert df_clean[pd.isnull(df_clean).any(axis=1)].shape[0] == 0, "Some rows ha
```

In [16]:
```
# Output dataset
df_clean.to_csv("data/output_fe.csv", index=False)
```

In [ ]:
```

```