shane_CS504_feature_engineering

December 3, 2024

1 Feature Engineering

It is apparent in the data structures in the FHFA dataset that the data are presented in a way that requires significant manipulations to prepare it for modeling correctly. This script resolves issues presented by the data manipualtion such that modeling can occur.

[1]:
```
import pandas as pd
import numpy as np
from src.static import DATA_DIR
```

First the data must be read into the notebook and we perform ths step below

[2]:
```
# read in data
mapped_data = pd.read_csv(f'{DATA_DIR}/mapped_data.csv', na_values=['.','NaN',␣
 ↪'None'])
# sort values in a useful way
mapped_data.sort_values(by=['year', 'enterprise_flag', 'record_number'],␣
 ↪inplace=True)
# ead in the last 5 years of data
mapped_data = mapped_data[mapped_data.year > (2023-5)]
```

[3]:
```
mapped_data[['record_number', 'enterprise_flag', 'date_of_mortgage_note',␣
 ↪'purpose_of_loan',
             'type_of_seller', 'federal_guarantee']].head(15)
```

[3]:

	record_number	enterprise_flag	date_of_mortgage_note	purpose_of_loan	\
291016	1	fannie	prior to year aquired	purchase	
291017	1	fannie	prior to year aquired	purchase	
291018	1	fannie	prior to year aquired	purchase	
291019	1	fannie	prior to year aquired	purchase	
291020	1	fannie	prior to year aquired	purchase	
291021	2	fannie	prior to year aquired	refinance	
291022	2	fannie	prior to year aquired	refinance	
291023	3	fannie	prior to year aquired	purchase	
291024	3	fannie	prior to year aquired	purchase	
291025	3	fannie	prior to year aquired	purchase	
291026	3	fannie	prior to year aquired	purchase	
291027	3	fannie	prior to year aquired	purchase	

```
291028              4          fannie  prior to year aquired       refinance
291029              5          fannie  prior to year aquired       refinance
291030              6          fannie  prior to year aquired       refinance

          type_of_seller federal_guarantee
291016  mortgage_company                no
291017  mortgage_company                no
291018  mortgage_company                no
291019  mortgage_company                no
291020  mortgage_company                no
291021  mortgage_company                no
291022  mortgage_company                no
291023  mortgage_company                no
291024  mortgage_company                no
291025  mortgage_company                no
291026  mortgage_company                no
291027  mortgage_company                no
291028  mortgage_company                no
291029  mortgage_company                no
291030  mortgage_company                no
```

[4]:
```
pd.get_dummies(mapped_data[['record_number', 'enterprise_flag']],␣
↪columns=['enterprise_flag']).head(15)
```

[4]:
```
        record_number  enterprise_flag_fannie  enterprise_flag_freddie
291016              1                    True                    False
291017              1                    True                    False
291018              1                    True                    False
291019              1                    True                    False
291020              1                    True                    False
291021              2                    True                    False
291022              2                    True                    False
291023              3                    True                    False
291024              3                    True                    False
291025              3                    True                    False
291026              3                    True                    False
291027              3                    True                    False
291028              4                    True                    False
291029              5                    True                    False
291030              6                    True                    False
```

[5]:
```
example = pd.DataFrame({'Red': [1, 0 , 0], 'Green': [0,1,0], 'Blue': [0,0,1]})
```

[6]:
```
example.T
```

[6]:
```
       0  1  2
Red    1  0  0
```

```
Green  0  1  0
Blue   0  0  1
```

[7]:
```
mapped_data[['record_number', 'num_units', 'num_bedrooms',␣
 ↪'affordability_level']].tail(15)
```

[7]:

	record_number	num_units	num_bedrooms	affordability_level
633435	2596	2.0	>=2	>50, <=60%
633436	2596	2.0	0-1	>=0, <=50%
633437	2596	1.0	>=2	>50, <=60%
633438	2596	1.0	>=2	>=0, <=50%
633439	2596	3.0	>=2	>60, <=80%
633440	2596	4.0	>=2	>=0, <=50%
633441	2596	25.0	>=2	>=0, <=50%
633442	2596	1.0	0-1	>=0, <=50%
633443	2597	44.0	>=2	>50, <=60%
633444	2597	16.0	>=2	>=0, <=50%
633445	2597	9.0	>=2	>=0, <=50%
633446	2597	4.0	>=2	>=0, <=50%
633447	2597	3.0	>=2	>=0, <=50%
633448	2597	2.0	>=2	>=0, <=50%
633449	2597	1.0	>=2	>=0, <=50%

The variables of num_bedrooms and affordability level contain the information that needs to be brough to the surface. The first step is going to be one hot encoding the values of these into their own columns so that we can map unit counts for each individual loan record over them. Then when we get the aggregate some of these per property per year we will have unit counts for each bedroom count and affordability level per property per year. We will use these values to predict on later.

[8]:
```
# encode columns
df = pd.get_dummies(
    mapped_data,
    columns=['num_bedrooms', 'affordability_level']
    )

# clean up new column names
df.columns = df.columns.str.strip()
df.columns
```

[8]:
```
Index(['year', 'enterprise_flag', 'record_number', 'census_tract_2020',
       'tract_income_ratio', 'affordability_cat', 'date_of_mortgage_note',
       'purpose_of_loan', 'type_of_seller', 'federal_guarantee',
       'tot_num_units', 'num_units', 'tenant_income_ind', 'num_bedrooms_0-1',
       'num_bedrooms_>=2', 'affordability_level_>100%',
       'affordability_level_>50, <=60%', 'affordability_level_>60, <=80%',
       'affordability_level_>80, <=100%', 'affordability_level_>=0, <=50%'],
      dtype='object')
```

Next we need to map the `num_units` for each record over each of the newly created columns so that when we aggregate the sum later we will get accurate counts

[9]:
```
# define a helper function
def unit_count_transformer(df: pd.DataFrame, cols=list[str]) -> pd.DataFrame:
    '''
    map unit counts to certain columns. needs at least one column named␣
 ↪`num_units` which is the
        target of the transformation. i.e., values from `num_units` are mapped␣
 ↪to columns in `cols`
        arg.
    arguments:
        df: a dataframe of data needing to be transformed
        cols: a list of specific column names that need to be worked on
    returns:
        a transformed dataframe
    '''
    # first create a copy so we arent working on the input dataframe
    output = df.copy()
    for col in cols:
        # map the number of units in each loan record to the value of each␣
 ↪input column
        output[col] = output.index.map(
            lambda x: output.loc[x]['num_units'] if output[col].loc[x] else 0
            )
    return output

# execute the transformation
df = unit_count_transformer(
    df, ['num_bedrooms_0-1', 'num_bedrooms_>=2', 'affordability_level_>100%',
         'affordability_level_>50, <=60%', 'affordability_level_>60, <=80%',
         'affordability_level_>80, <=100%', 'affordability_level_>=0, <=50%']
    )
```

[10]:
```
df[['num_units', 'num_bedrooms_0-1', 'num_bedrooms_>=2',␣
 ↪'affordability_level_>100%',
    'affordability_level_>50, <=60%','affordability_level_>60, <=80%',
    'affordability_level_>80, <=100%','affordability_level_>=0, <=50%']].
 ↪tail(15)
```

[10]:

	num_units	num_bedrooms_0-1	num_bedrooms_>=2	\
633435	2.0	0.0	2.0	
633436	2.0	2.0	0.0	
633437	1.0	0.0	1.0	
633438	1.0	0.0	1.0	
633439	3.0	0.0	3.0	
633440	4.0	0.0	4.0	

633441	25.0	0.0	25.0
633442	1.0	1.0	0.0
633443	44.0	0.0	44.0
633444	16.0	0.0	16.0
633445	9.0	0.0	9.0
633446	4.0	0.0	4.0
633447	3.0	0.0	3.0
633448	2.0	0.0	2.0
633449	1.0	0.0	1.0

	affordability_level_>100%	affordability_level_>50, <=60%	\
633435	0.0	2.0	
633436	0.0	0.0	
633437	0.0	1.0	
633438	0.0	0.0	
633439	0.0	0.0	
633440	0.0	0.0	
633441	0.0	0.0	
633442	0.0	0.0	
633443	0.0	44.0	
633444	0.0	0.0	
633445	0.0	0.0	
633446	0.0	0.0	
633447	0.0	0.0	
633448	0.0	0.0	
633449	0.0	0.0	

	affordability_level_>60, <=80%	affordability_level_>80, <=100%	\
633435	0.0	0.0	
633436	0.0	0.0	
633437	0.0	0.0	
633438	0.0	0.0	
633439	3.0	0.0	
633440	0.0	0.0	
633441	0.0	0.0	
633442	0.0	0.0	
633443	0.0	0.0	
633444	0.0	0.0	
633445	0.0	0.0	
633446	0.0	0.0	
633447	0.0	0.0	
633448	0.0	0.0	
633449	0.0	0.0	

	affordability_level_>=0, <=50%
633435	0.0
633436	2.0

```
633437                                0.0
633438                                1.0
633439                                0.0
633440                                4.0
633441                               25.0
633442                                1.0
633443                                0.0
633444                               16.0
633445                                9.0
633446                                4.0
633447                                3.0
633448                                2.0
633449                                1.0
```

Finally we perform a massive grouping and aggregation. Rows which have the same `record_number` have columns which are *always* the same value within that same `record_number`. For example, given `record_number == 1` for a given `year` and `enterprise_flag`. That is to say, each of these individual record nubmer, year, enterprise flag combinations may have a value for `date_of_morgage_note` which does not vary despute multiple entries in our dataset for that combined index. These columns are identified and included in the grouping statement below because they are ingtegral to one record.

[11]:
```
# this multistage grouping and aggregation creates 1 record with counts of␣
 ↪units in certain columns
df = df.groupby(
    # define grouping columns for record grouping
    ['year', 'enterprise_flag', 'record_number', 'census_tract_2020',␣
 ↪'tract_income_ratio',
     'date_of_mortgage_note', 'purpose_of_loan', 'type_of_seller',␣
 ↪'federal_guarantee',
     'tenant_income_ind', 'affordability_cat', 'tot_num_units']
    # this next step identifies which columns we're going to sum up
    )[['num_units', 'num_bedrooms_0-1', 'num_bedrooms_>=2',␣
 ↪'affordability_level_>100%',
       'affordability_level_>50, <=60%', 'affordability_level_>60, <=80%',
       'affordability_level_>80, <=100%', 'affordability_level_>=0, <=50%']].
 ↪agg('sum').reset_index()

print('Data aggregation yields a DataFrame containing aggregate counts of␣
 ↪certain categories ',
      df.head())
```

```
Data aggregation yields a DataFrame containing aggregate counts of certain
categories     year enterprise_flag  record_number census_tract_2020
tract_income_ratio  \
0  2019          fannie              1       >=30% <100%          >0, <=80%
1  2019          fannie              2       >=30% <100%        >10, <=120%
2  2019          fannie              3              <10%              >120%
```

```
3  2019          fannie              8       >=30% <100%        >10, <=120%
4  2019          fannie             10       >=30% <100%        >10, <=120%

   date_of_mortgage_note purpose_of_loan    type_of_seller federal_guarantee  \
0  prior to year aquired        purchase  mortgage_company                no
1  prior to year aquired       refinance  mortgage_company                no
2  prior to year aquired        purchase  mortgage_company                no
3  prior to year aquired       refinance  mortgage_company                no
4  prior to year aquired       refinance  mortgage_company                no

  tenant_income_ind affordability_cat tot_num_units  num_units  \
0                No      >=20%, >=40%         25-50        5.0
1                No        <20%, <40%         25-50        2.0
2                No        <20%, <40%         51-99        5.0
3                No      >=20%, >=40%         25-50        2.0
4                No        <20%, <40%       100-149        4.0

   num_bedrooms_0-1  num_bedrooms_>=2  affordability_level_>100%  \
0               2.0               3.0                        0.0
1               0.0               2.0                        0.0
2               2.0               3.0                        1.0
3               2.0               0.0                        0.0
4               2.0               2.0                        2.0

   affordability_level_>50, <=60%  affordability_level_>60, <=80%  \
0                             2.0                             1.0
1                             1.0                             1.0
2                             0.0                             2.0
3                             0.0                             1.0
4                             0.0                             1.0

   affordability_level_>80, <=100%  affordability_level_>=0, <=50%
0                              0.0                             2.0
1                              0.0                             0.0
2                              2.0                             0.0
3                              0.0                             1.0
4                              1.0                             0.0
```

[12]:
```
# finally prepare remaining categorical columns for modeling by finishing one␣
 ↪hot encoding
df = pd.get_dummies(
    df,
    columns=['enterprise_flag', #'census_tract_2020', # commented out to leave␣
 ↪as a categorical for testing ordinal regressors.
             #'tract_income_ratio', # commented out to leave as categorical for␣
 ↪testing ordinal variable
             'date_of_mortgage_note',
```

```
            'purpose_of_loan', 'type_of_seller', 'federal_guarantee',␣
↪'tenant_income_ind',
            #'affordability_cat', # commented out to leave as a categorical␣
↪for testing ordinal regressors.
            'tot_num_units'],
   drop_first = True
   )
```

This step is necessary to examine the these two variables as ordinal values as they were originally provided

[13]:
```
census_tract_2020 = {'<10%': '1', '>=10%, <30%': '2', '>=30% <100%': '3', 'NaN':
↪ '9'}
affordability_cat = {'>=20%, <40%': '1', '<20%, >=40%': '2', '>=20%, >=40%':␣
↪'3',
                     '<20%, <40%': '4'}
tract_income_ratio, = {'>0, <=80%': '1', '>10, <=120%': '2', '>120%': '3'},

# remap ordinal values over the string values for modeling
df.census_tract_2020 = df.census_tract_2020.map(lambda x: census_tract_2020.
↪get(x))
df.affordability_cat = df.affordability_cat.map(lambda x: affordability_cat.
↪get(x))
df.tract_income_ratio = df.tract_income_ratio.map(lambda x: tract_income_ratio.
↪get(x))
```

Finally, since our data encapsulates the onset of COVID a very macro influential global event, it may be prudent to study what signal can be derived from a feature that encodes wether a record is pre or post covid

[14]:
```
# create simple flag to tell the model about covid
df['after_covid_ind'] = df.year >= 2020
df.columns = df.columns.str.strip().str.replace(' - ', '-')
```

[ ]:
```
# create simple count of number of affordable units so that a predictor can␣
↪predict the number of affordabile units based on other inputs.
df['num_affordable_units'] = df[['affordability_level_>=0, <=50%',␣
↪'affordability_level_>50, <=60%'
                                 , 'affordability_level_>60, <=80%'
                                ]].sum(axis=1)
```

[16]:
```
df[['num_units', 'num_bedrooms_0-1', 'num_bedrooms_>=2',␣
↪'num_affordable_units']].tail(15)
```

[16]:

	num_units	num_bedrooms_0-1	num_bedrooms_>=2	num_affordable_units
34253	31.0	0.0	31.0	7.0
34254	59.0	9.0	50.0	59.0

34255	63.0	25.0	38.0	0.0
34256	24.0	17.0	7.0	4.0
34257	26.0	26.0	0.0	26.0
34258	6.0	5.0	1.0	1.0
34259	10.0	10.0	0.0	0.0
34260	120.0	27.0	93.0	1.0
34261	28.0	25.0	3.0	1.0
34262	19.0	8.0	11.0	19.0
34263	19.0	12.0	7.0	0.0
34264	32.0	31.0	1.0	16.0
34265	188.0	26.0	162.0	188.0
34266	125.0	6.0	119.0	122.0
34267	79.0	0.0	79.0	79.0

[17]:
```
# save engineered data
df.to_csv(f'{DATA_DIR}/preprocessed_data.csv', index=False)
## End script
```

[ ]:
```

```