2_Explore

May 3, 2021

[1]:
```
import modin.pandas as pd
import numpy as np

import pickle

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
import keras.backend as K
```

```
Using TensorFlow backend.
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
```

```
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
```

[2]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

[3]:
```
training_dat = pd.read_pickle('./PROCESSED/training_dat2.pkl')
training_dat.columns
```

```
UserWarning: Ray execution environment not yet initialized. Initializing…
To remove this warning, run the following python code before doing dataframe
operations:

    import ray
    ray.init()

UserWarning: `read_pickle` defaulting to pandas implementation.
To request implementation, send an email to feature_requests@modin.org.
```

[3]:
```
Index(['Loan Sequence Number', 'Loan Deliquency Within Year', 'Credit Score',
       'MortgageInsuranceFlag', 'Units_1', 'Units_2', 'Units_3', 'Units_4',
       'OccupancyStatus_I', 'OccupancyStatus_P', 'OccupancyStatus_S',
       'Original Combined Loan-to-Value (CLTV)',
       'Original Debt-to-Income (DTI) Ratio', 'Original UPB',
       'Original Loan-to-Value (LTV)', 'Original Interest Rate', 'Channel_B',
       'Channel_C', 'Channel_R', 'PropertyType_CO', 'PropertyType_CP',
       'PropertyType_MH', 'PropertyType_PU', 'PropertyType_SF',
       'LoanPurpose_C', 'LoanPurpose_N', 'LoanPurpose_P', 'LoanTerm_360',
       'LoanTerm_180', 'LoanTerm_240', 'LoanTerm_120', 'LoanTerm_300',
       'Original Loan Term', 'OneBorrower', 'AffordableProgramFlag'],
      dtype='object')
```

[6]:
```
with open("./PROCESSED/scalers.pkl","rb") as f:
    scaler_dct = pickle.load(f)
scaler_dct.keys()
```

```
[6]: dict_keys(['Credit Score', 'Original Combined Loan-to-Value (CLTV)', 'Original
Debt-to-Income (DTI) Ratio', 'Original UPB', 'Original Loan-to-Value (LTV)',
'Original Interest Rate', 'Original Loan Term'])
```

Converting scaled values back to original for analysis

```
[9]: scaled_cols = list(scaler_dct.keys())
for col in scaled_cols:
    scaler = scaler_dct[col]
    training_dat[col] = scaler.inverse_transform(training_dat[[col]])
training_dat.head()
```

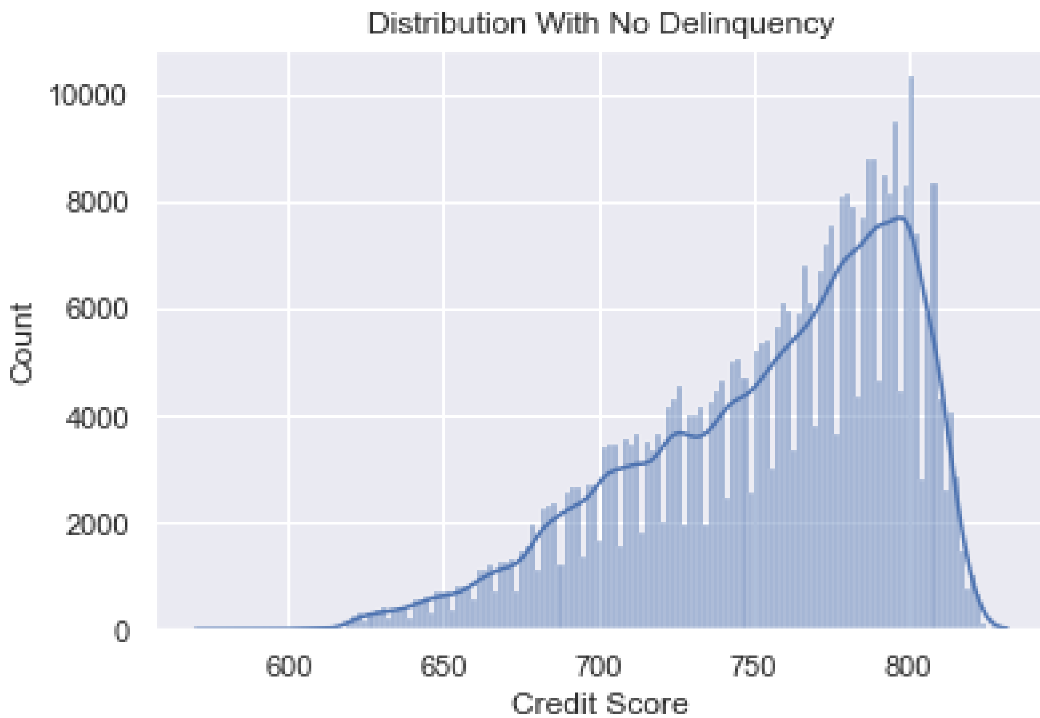
```
[9]:   Loan Sequence Number  Loan Deliquency Within Year  Credit Score  \
0         F110Q1000008                          0.0         804.0
1         F110Q1000064                          0.0         816.0
2         F110Q1000072                          0.0         783.0
3         F110Q1000080                          0.0         667.0
4         F110Q1000096                          0.0         799.0

   MortgageInsuranceFlag  Units_1  Units_2  Units_3  Units_4  \
0                      0        1        0        0        0
1                      0        1        0        0        0
2                      0        1        0        0        0
3                      0        1        0        0        0
4                      0        1        0        0        0

   OccupancyStatus_I  OccupancyStatus_P  …  LoanPurpose_N  LoanPurpose_P  \
0                  0                  1  …              1              0
1                  0                  1  …              0              0
2                  0                  1  …              1              0
3                  0                  1  …              1              0
4                  0                  1  …              1              0

   LoanTerm_360  LoanTerm_180  LoanTerm_240  LoanTerm_120  LoanTerm_300  \
0             1             0             0             0             0
1             1             0             0             0             0
2             1             0             0             0             0
3             1             0             0             0             0
4             1             0             0             0             0

   Original Loan Term  OneBorrower  AffordableProgramFlag
0               360.0            0                      0
1               360.0            1                      0
2               360.0            0                      0
3               360.0            1                      0
4               360.0            1                      0
```

[5 rows x 35 columns]

0.1 Compare credit scores of non delinquent vs delinquent owners

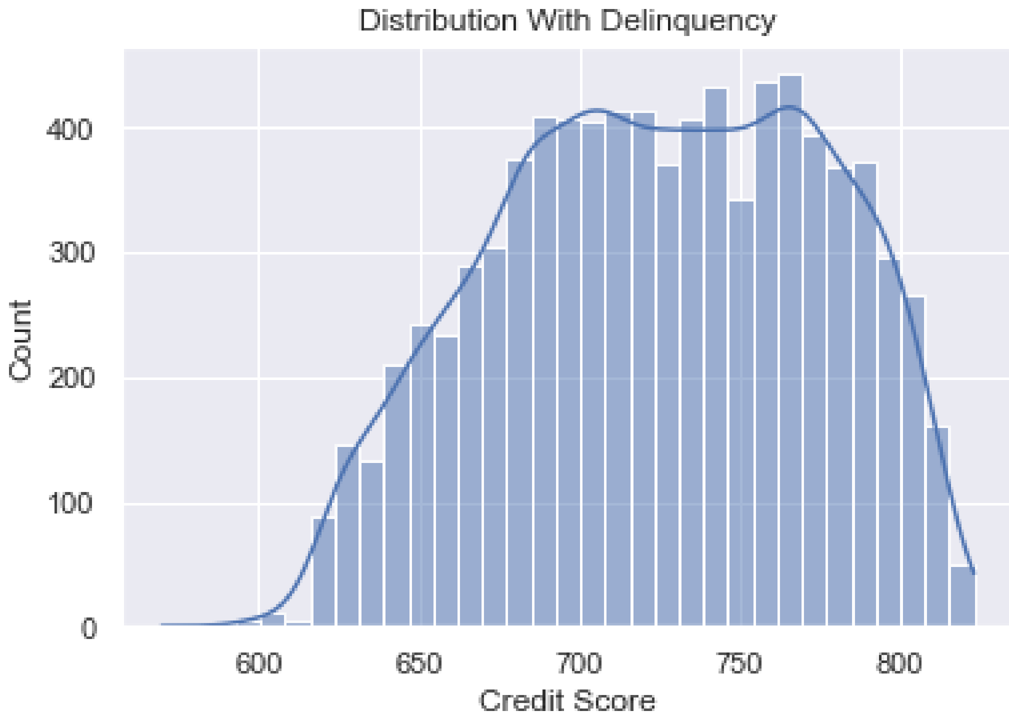
[13]:
```
tf = training_dat['Loan Deliquency Within Year'] == 0
ax = sns.histplot(x="Credit Score", data=training_dat[tf], kde=True)
ax.set_title("Distribution With No Delinquency")
```

[13]: Text(0.5, 1.0, 'Distribution With No Delinquency')



[14]:
```
tf = training_dat['Loan Deliquency Within Year'] == 1
ax = sns.histplot(x="Credit Score", data=training_dat[tf], kde=True)
ax.set_title("Distribution With Delinquency")
```

[14]: Text(0.5, 1.0, 'Distribution With Delinquency')



Buyers that suffer a loan delinquency within a year tend are more likely to have lower credit scores then the general loan population.

```
[ ]:
```

0.2 Checkout Mortgage Insurance Percentage (MI %)

```
[20]: mi_gp = training_dat.groupby('Mortgage Insurance Percentage (MI %)')
      mi_dat = mi_gp.agg({'Loan Deliquency Within Year':'mean',
                          'Loan Sequence Number':'count'})
      # only look at buckets with more than 1000 loans
      tf = mi_dat['Loan Sequence Number'] > 1000
      mi_dat[tf]
```

```
[20]:                                       Loan Deliquency Within Year  \
      Mortgage Insurance Percentage (MI %)
      0.0                                                      0.020821
      6.0                                                      0.012256
      12.0                                                     0.024241
      25.0                                                     0.022132
      30.0                                                     0.023472

                                            Loan Sequence Number
```

```
Mortgage Insurance Percentage (MI %)
0.0                                                        309585
6.0                                                          1795
12.0                                                        11922
25.0                                                        28827
30.0                                                        43541
```

Change this to boolean feature, since % value doesn't seem significant

[21]:
```
training_dat['No Mortgage Insurance'] = 1

tf = training_dat['Mortgage Insurance Percentage (MI %)'] > 0
training_dat.loc[tf, 'No Mortgage Insurance'] = 0
```

[23]:
```
training_dat.groupby('No Mortgage Insurance')['Loan Deliquency Within Year'].
 ↪mean()
```

[23]:
```
No Mortgage Insurance
0    0.022946
1    0.020821
Name: Loan Deliquency Within Year, dtype: float64
```

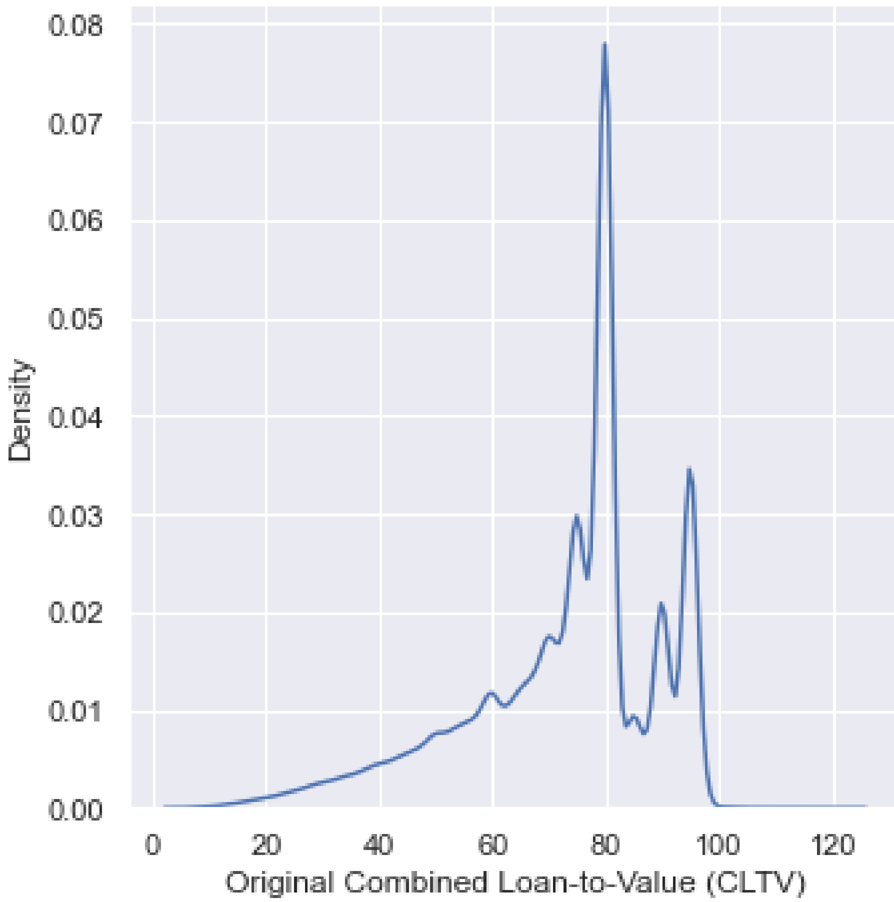
There is a stronger likelyhood of delinquency on loans with mortgage insurance.

[ ]:
```

```

0.3 Compare CLTV of non delinquent vs delinquent owners

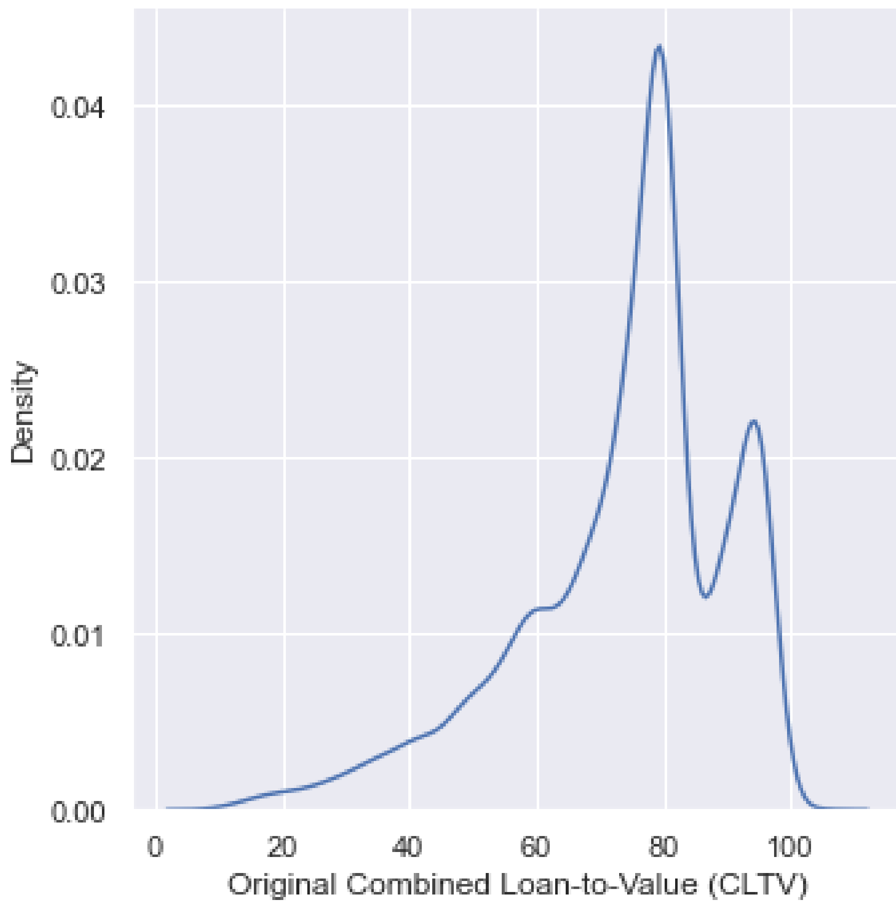
[32]:
```
tf = training_dat['Loan Deliquency Within Year'] == 0
sns.displot(x="Original Combined Loan-to-Value (CLTV)", data=training_dat[tf],␣
 ↪kind='kde')
```

[32]:
```
<seaborn.axisgrid.FacetGrid at 0x2a8eb251048>
```



[33]:
```
tf = training_dat['Loan Deliquency Within Year'] == 1
sns.displot(x="Original Combined Loan-to-Value (CLTV)", data=training_dat[tf],␣
 ↪kind='kde')
```

[33]: <seaborn.axisgrid.FacetGrid at 0x2a8ec3fc0f0>



The CLTVs look identitical between these groups

[ ]:

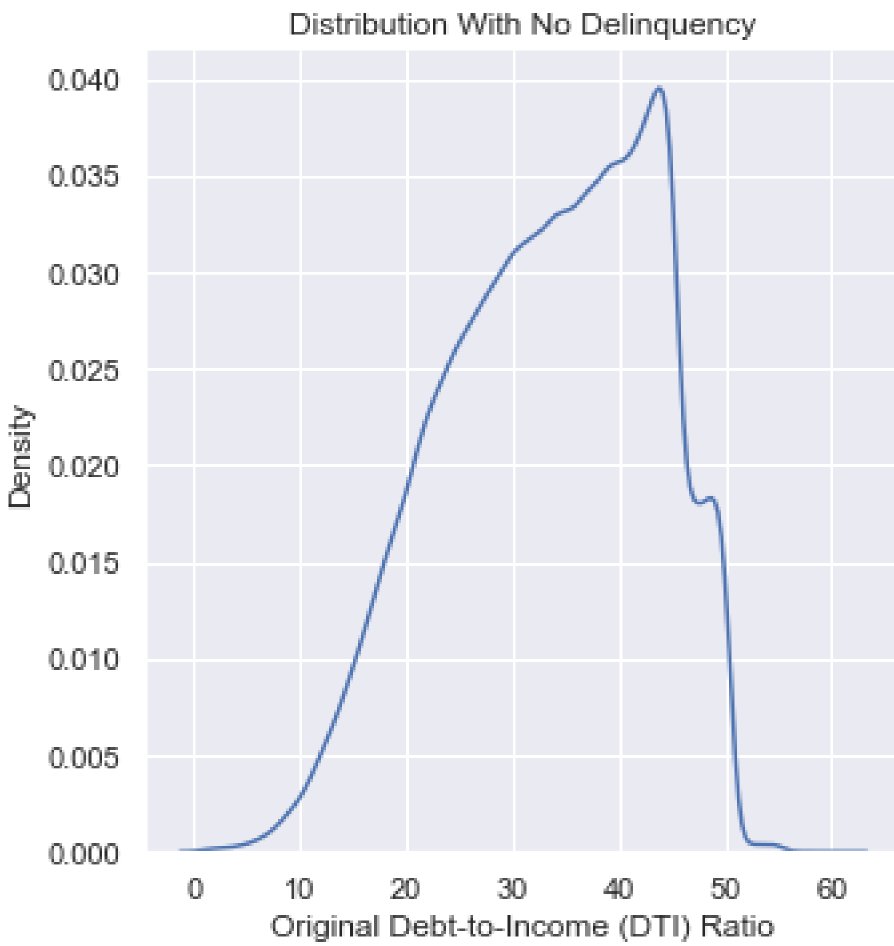
```

```

0.4 Compare DTI of non delinquent vs delinquent owners

[23]:

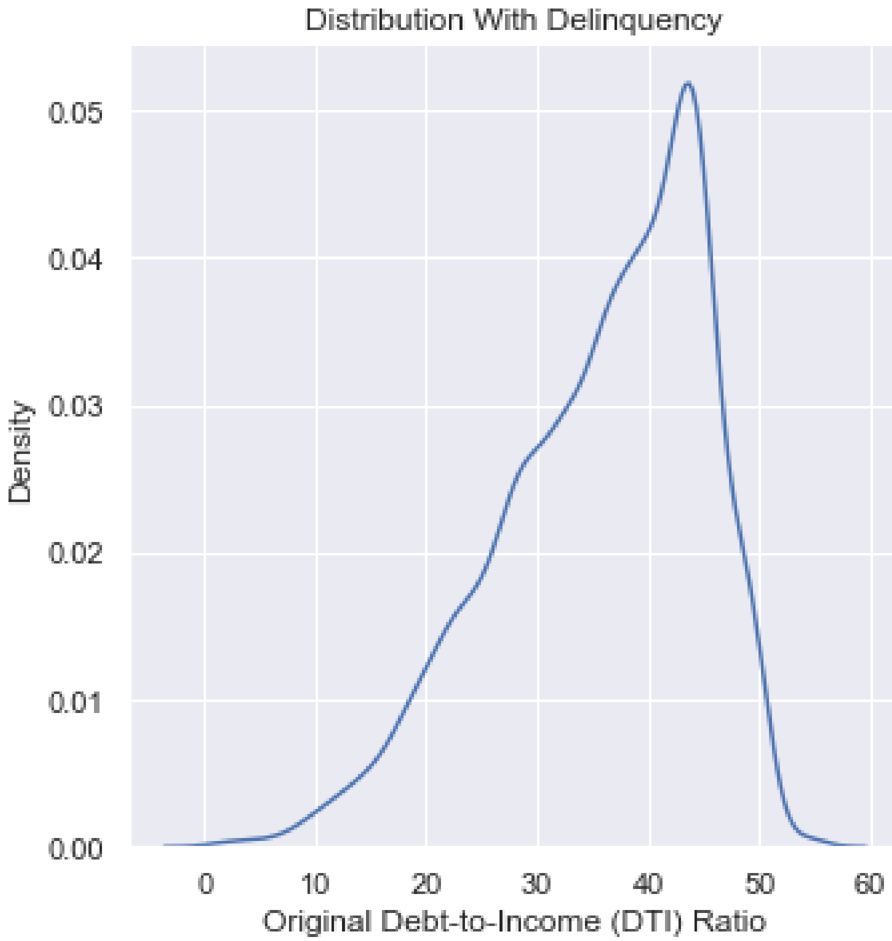
```
tf = training_dat['Loan Deliquency Within Year'] == 0
sns.displot(x="Original Debt-to-Income (DTI) Ratio", data=training_dat[tf],␣
 ↪kind='kde')
plt.title("Distribution With No Delinquency")
```

[23]: Text(0.5, 1.0, 'Distribution With No Delinquency')



[24]:
```
tf = training_dat['Loan Deliquency Within Year'] == 1
ax = sns.displot(x="Original Debt-to-Income (DTI) Ratio",␣
 ↪data=training_dat[tf], kind='kde')
plt.title("Distribution With Delinquency")
```

[24]:
```
Text(0.5, 1.0, 'Distribution With Delinquency')
```

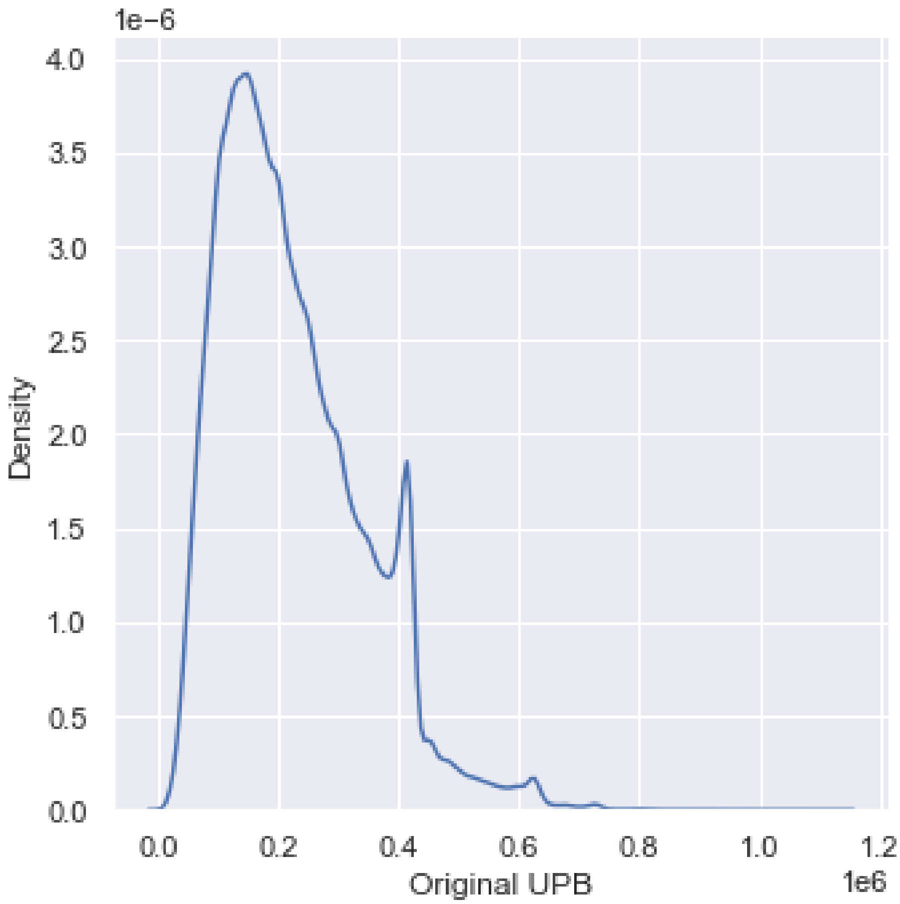


Although both populations have a peak at 40% DTI, more of the delinquent borrowers were at that peak.

0.5 Compare UPB of non delinquent vs delinquent owners

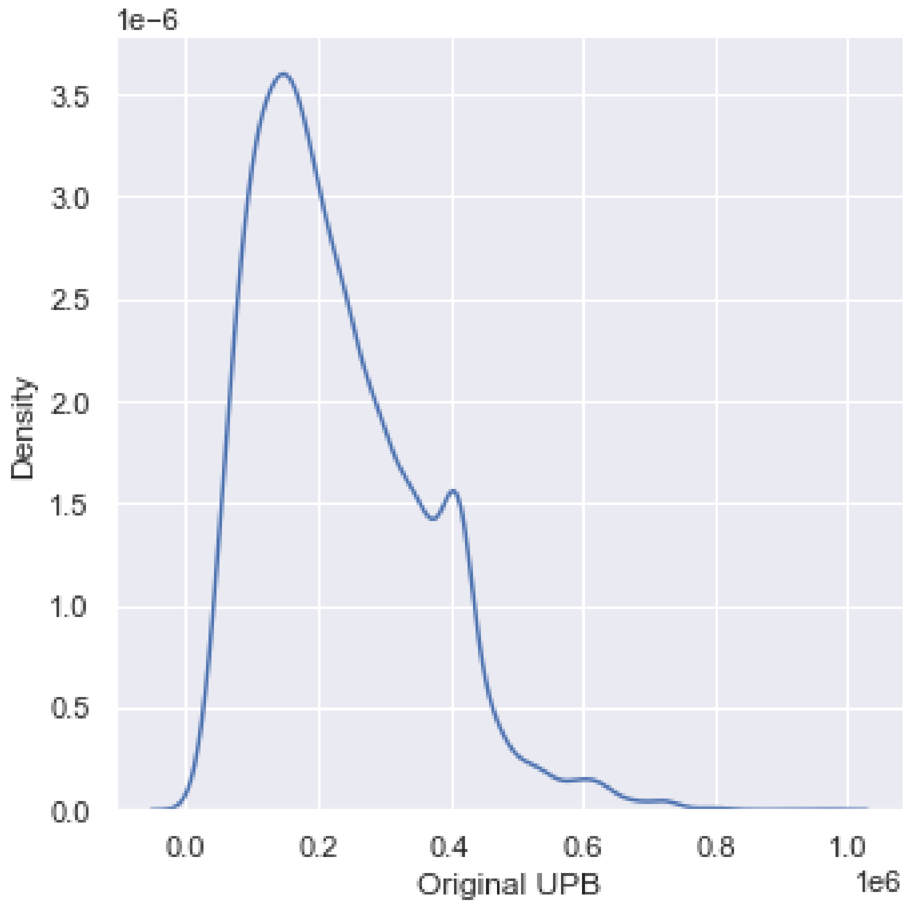
[36]:
```
tf = training_dat['Loan Deliquency Within Year'] == 0
sns.displot(x="Original UPB", data=training_dat[tf], kind='kde')
```

[36]:
```
<seaborn.axisgrid.FacetGrid at 0x2a8eb251898>
```



[37]:
```
tf = training_dat['Loan Deliquency Within Year'] == 1
sns.displot(x="Original UPB", data=training_dat[tf], kind='kde')
```

[37]: <seaborn.axisgrid.FacetGrid at 0x2a8ed916c18>

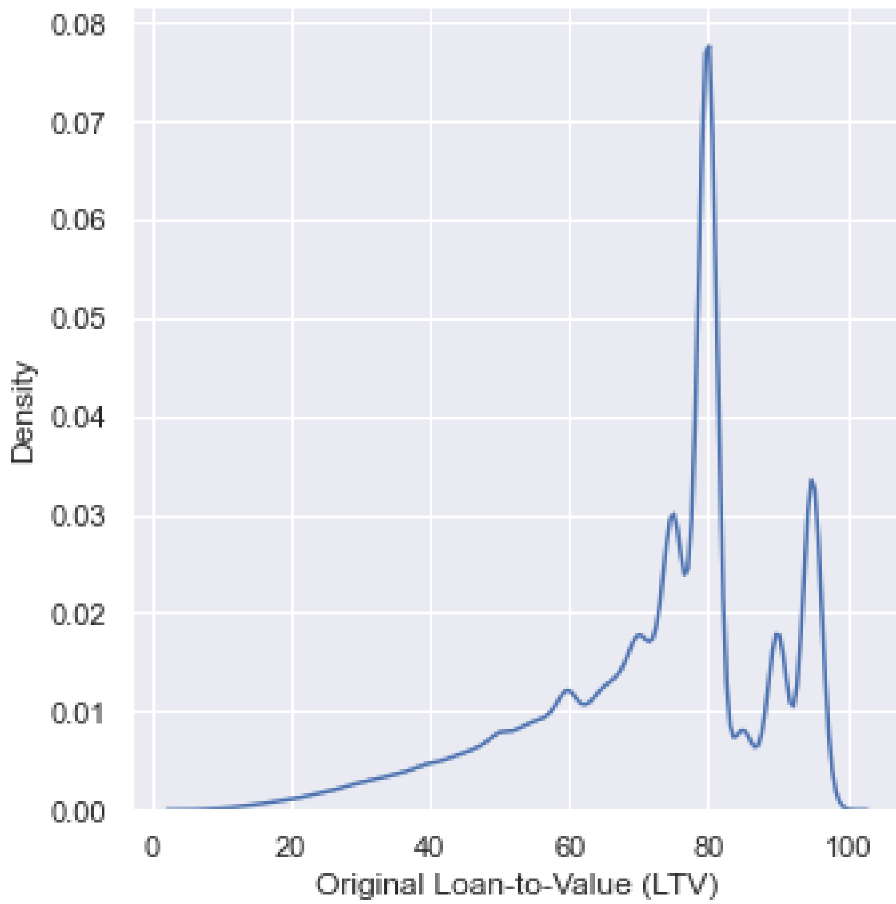


Both of these distributions look identical

0.6 Compare LTV of non delinquent vs delinquent owners

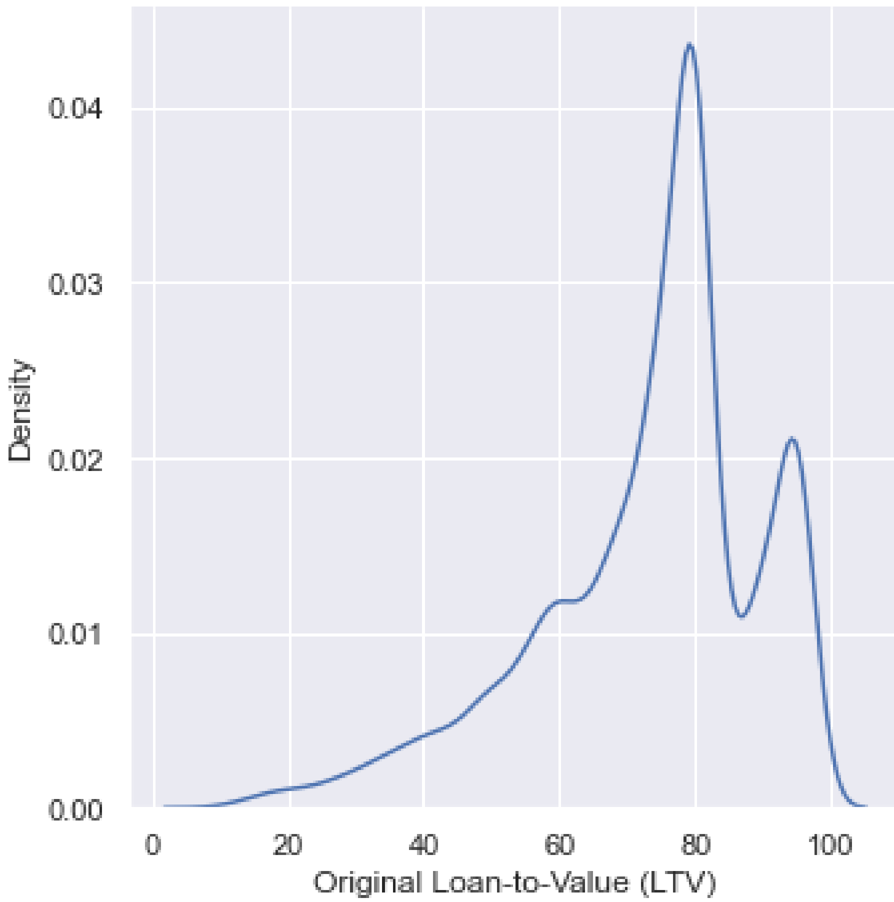
[38]:
```
tf = training_dat['Loan Deliquency Within Year'] == 0
sns.displot(x="Original Loan-to-Value (LTV)", data=training_dat[tf], kind='kde')
```

[38]:
```
<seaborn.axisgrid.FacetGrid at 0x2a8ed8f76a0>
```



[39]:
```
tf = training_dat['Loan Deliquency Within Year'] == 1
sns.displot(x="Original Loan-to-Value (LTV)", data=training_dat[tf], kind='kde')
```

[39]: <seaborn.axisgrid.FacetGrid at 0x2a8ed8f7518>



The groups look identical here as well. This feature also looks identical to CLTV, so I should remove one of them from what is fed to the model.

0.7 Compare Original Interest Rate of non delinquent vs delinquent owners

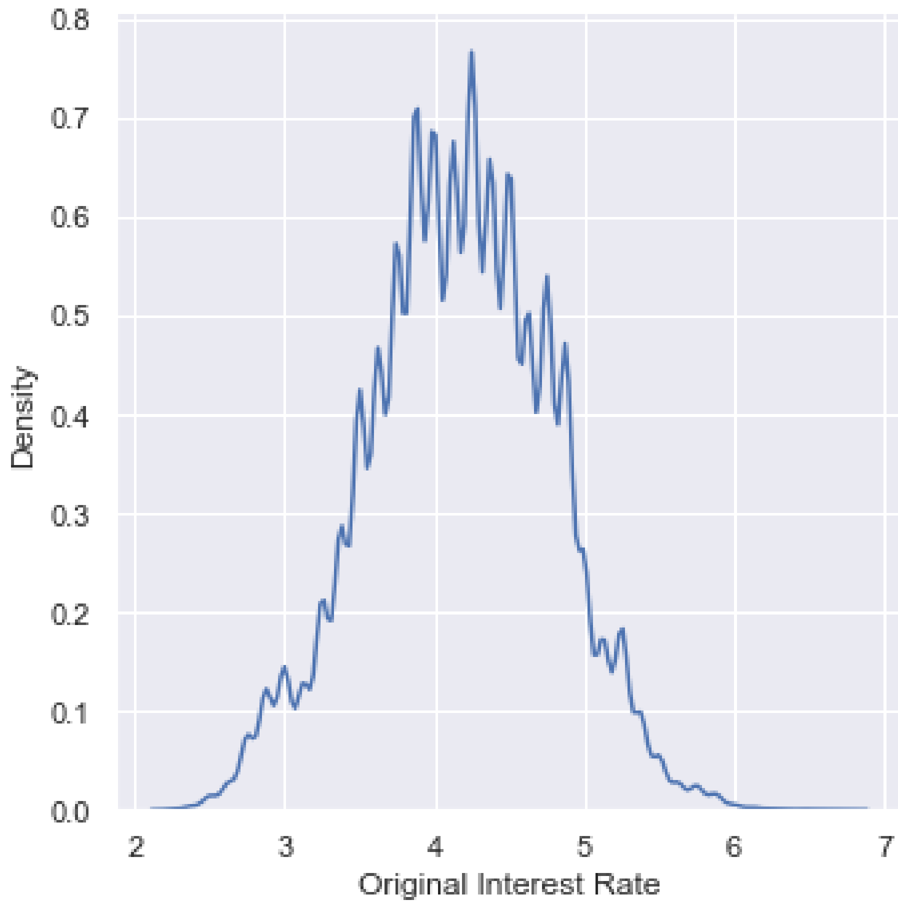
[32]:

```
```

[46]:

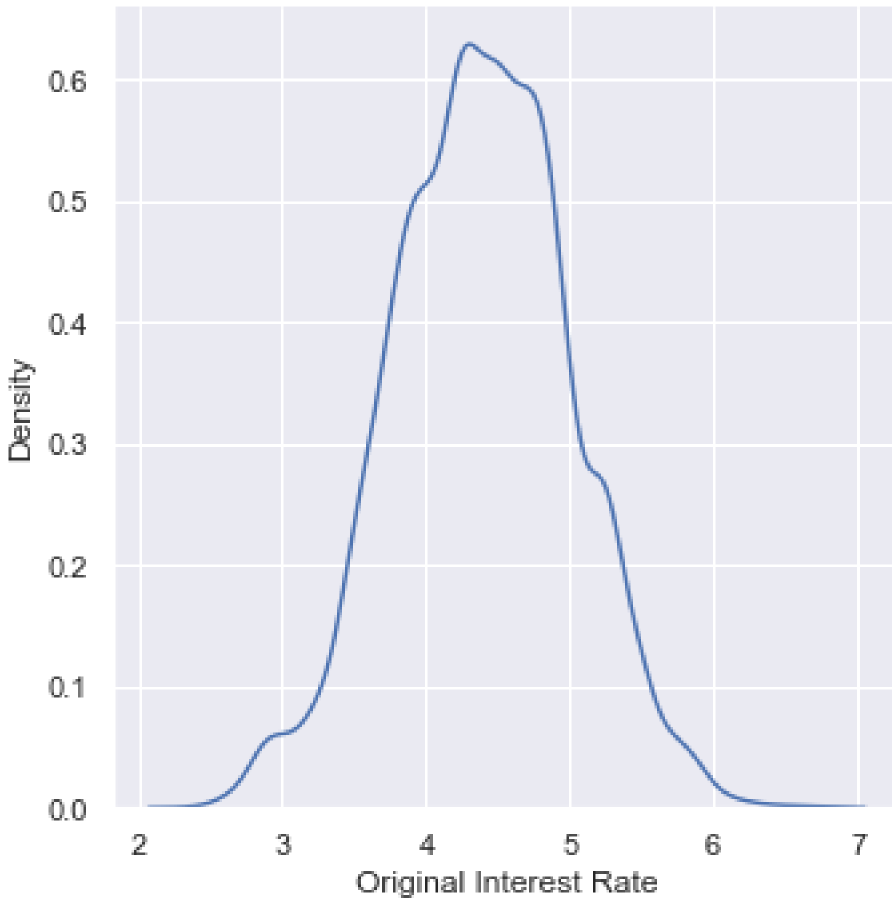
```
tf = training_dat['Loan Deliquency Within Year'] == 0
sns.displot(x="Original Interest Rate", data=training_dat[tf], kind='kde')
```

[46]: <seaborn.axisgrid.FacetGrid at 0x2a8edcce6d8>



[45]:
```
tf = training_dat['Loan Deliquency Within Year'] == 1
sns.displot(x="Original Interest Rate", data=training_dat[tf], kind='kde')
```

[45]: <seaborn.axisgrid.FacetGrid at 0x2a8eda76748>



Although the distributions are similar, it does appear like the interest rates are higher for delinquent loans.

0.8 Loan Term Comparison

[54]:
```
lt_gp = training_dat.groupby('Original Loan Term')
lt_dat = lt_gp.agg({'Loan Deliquency Within Year':'mean',
                    'Loan Sequence Number':'count'})
# only look at buckets with more than 1000 loans
tf = lt_dat['Loan Sequence Number'] > 1000
lt_dat[tf]
```

[54]:

Original Loan Term	Loan Deliquency Within Year	Loan Sequence Number
120.0	0.008825	6232
180.0	0.015296	74331
240.0	0.016093	18952
300.0	0.017710	2428

```
360.0                                 0.023552                292243
```

Not what I expected. Longer loan terms have higher likelyhood of delinquency

[ ]:

```

```

0.9 Binary flags

[58]:

```python
binary_cols = ['Units_1', 'Units_2', 'Units_3', 'Units_4',
               'OccupancyStatus_I', 'OccupancyStatus_P', 'OccupancyStatus_S',
               'Channel_B', 'Channel_C', 'Channel_R',
               'PropertyType_CO', 'PropertyType_CP', 'PropertyType_MH',␣
 ↪'PropertyType_PU', 'PropertyType_SF',
                'LoanPurpose_C', 'LoanPurpose_N', 'LoanPurpose_P',
               'LoanTerm_360', 'LoanTerm_180', 'LoanTerm_240', 'LoanTerm_120',␣
 ↪'LoanTerm_300',
               'OneBorrower', 'AffordableProgramFlag']
for col in binary_cols:
    tf = training_dat[col] == 1
    drat = training_dat[tf]['Loan Deliquency Within Year'].sum() /␣
 ↪training_dat[col].sum()

    print("Delinquency rate for " + str(col) + ": " + str(drat))
    print(str(col) + " count: " + str(training_dat[col].sum()))

drat = training_dat['Loan Deliquency Within Year'].sum() / len(training_dat)
print("Overall delinquency: " + str(drat))
print("Total count: " + str(len(training_dat)))
```

```
Delinquency rate for Units_1: 0.021091811414392061
Units_1 count: 388492
Delinquency rate for Units_2: 0.028799702712746192
Units_2 count: 5382
Delinquency rate for Units_3: 0.042502004811547714
Units_3 count: 1247
Delinquency rate for Units_4: 0.028795811518324606
Units_4 count: 1146
Delinquency rate for OccupancyStatus_I: 0.0292701646664806
OccupancyStatus_I count: 28664
Delinquency rate for OccupancyStatus_P: 0.020531617665906235
OccupancyStatus_P count: 351117
Delinquency rate for OccupancyStatus_S: 0.02347446318088075
OccupancyStatus_S count: 16486
Delinquency rate for Channel_B: 0.025317046967326062
Channel_B count: 42817
Delinquency rate for Channel_C: 0.02512746972594009
Channel_C count: 125520
```

```
Delinquency rate for Channel_R: 0.01841354801912868
Channel_R count: 227930
Delinquency rate for PropertyType_CO: 0.02314644720405614
PropertyType_CO count: 27218
Delinquency rate for PropertyType_CP: 0.01694915254237288
PropertyType_CP count: 531
Delinquency rate for PropertyType_MH: 0.018991964937910884
PropertyType_MH count: 1369
Delinquency rate for PropertyType_PU: 0.019564827187331723
PropertyType_PU count: 100282
Delinquency rate for PropertyType_SF: 0.021763650057894008
PropertyType_SF count: 266867
Delinquency rate for LoanPurpose_C: 0.024252927892442624
LoanPurpose_C count: 95205
Delinquency rate for LoanPurpose_N: 0.017350295733734644
LoanPurpose_N count: 114292
Delinquency rate for LoanPurpose_P: 0.022182363334582643
LoanPurpose_P count: 186770
Delinquency rate for LoanTerm_360: 0.023552317762957537
LoanTerm_360 count: 292243
Delinquency rate for LoanTerm_180: 0.015296444283004399
LoanTerm_180 count: 74331
Delinquency rate for LoanTerm_240: 0.01609328830730266
LoanTerm_240 count: 18952
Delinquency rate for LoanTerm_120: 0.008825417201540436
LoanTerm_120 count: 6232
Delinquency rate for LoanTerm_300: 0.01771004942339374
LoanTerm_300 count: 2428
Delinquency rate for OneBorrower: 0.027290618849770392
OneBorrower count: 182920
Delinquency rate for AffordableProgramFlag: 0.02989627821842587
AffordableProgramFlag count: 3278
Overall delinquency: 0.021286153023087968
Total count: 396267
```

I should get rid of PropertyType_CP due to few observations, but otherwise these flags have valuable information.

[ ]:
```

```

[ ]:
```

```