Consumer Loans

October 14, 2018

1 Consumer Loans

It may be possible to predict the default probability of consumer loans and investigate which variables specific to the loan is a significant predictor. In addition, we may be able to determine the value of a consumer if they have defaulted. We therefore have two tasks: determine which variables have the most significant impact on default probability; obtain a model for the value of a consumer once they have defaulted.

For variable significance and variable selection, we can start with a linear model. But first we explore the data once we have imported the necessary libraries.

1.1 Import Libraries

```
In [1]: import sys
        import os
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import math
        import statistics
        import datetime
        import matplotlib
        %matplotlib inline
```

```
In [2]: from sklearn.linear_model import LogisticRegression
        from sklearn.linear_model import LinearRegression
        import statsmodels.api as sm
        from scipy import stats
        import sklearn
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import classification_report, accuracy_score
        from sklearn.metrics import confusion_matrix
```

```
C:\Users\HVAD\Anaconda3\lib\site-packages\statsmodels\compat\pandas.py:56: FutureWarning: The p
  from pandas.core import datetools
```

```
In [3]: # Get Versions
        print('Python Version = {}'.format(sys.version[0:31]))
        print('Numpy Version = {}'.format(np.__version__))
        print('Pandas Version = {}'.format(pd.__version__))
        print('Matplotlib Version = {}'.format(matplotlib.__version__))
        print('Seaborn Version = {}'.format(sns.__version__))
        print('sklearn Version = {}'.format(sklearn.__version__))
```

```
Python Version = 3.6.4 |Anaconda custom (64-bit)
Numpy Version = 1.14.0
Pandas Version = 0.22.0
Matplotlib Version = 2.1.2
Seaborn Version = 0.8.1
sklearn Version = 0.19.1
```

1.2 The Loans Dataset

First we read in the excel sheets and write to csv as a 1-time action since reading from CSV is quicker in subsequent notebook startups.

```
In [4]: if((not os.path.isfile('Performance.csv')) or (not os.path.isfile('StaticData.csv'))):
            StaticData = pd.read_excel('LoansDataset.xlsx',sheet_name='Static Data')
            Performance = pd.read_excel('LoansDataset.xlsx',sheet_name='Performance',header=1)

            # Note that here, all NA values are converted to null or empty
            StaticData.to_csv('StaticData.csv',index=False)
            Performance.to_csv('Performance.csv',index=False)
```

```
In [5]: StaticData = pd.read_csv('StaticData.csv',low_memory=False)
        Performance = pd.read_csv('Performance.csv',low_memory=False)
```

We can see below that there are 72167 records. The LoanID is viable for use as a unique index since we see that there are 72167 unique values for this column. This also means that we can merge the StaticData DataFrame with the Performance DataFrame on LoanID and not generate multiple records.

```
In [6]: StaticData.info()

        print()
        print()

        print('The number of unique entries for LoanID in the StaticData DataFrame is {}'.forma

        print()
        print()

        print('The number of unique entries for LoanID in the Performance DataFrame is {}'.forn
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 72167 entries, 0 to 72166
Data columns (total 20 columns):
Year                              72167 non-null int64
LoanID                            72167 non-null int64
Original Loan Size                72167 non-null float64
Origination Date                  72167 non-null object
Original Term                     72167 non-null int64
APR                               72167 non-null float64
Loan Rate                         72167 non-null float64
LTV                               72167 non-null float64
Monthly Instalment                72167 non-null float64
Type 1                            72167 non-null object
Type 2                            72167 non-null object
Borrower's Employment Status      72167 non-null object
Risk Tier                         72167 non-null object
Exposure At Default               3168 non-null float64
Further Recovery Expected Y/N     3168 non-null object
Recoveries @ 6m                   1801 non-null float64
Recoveries @ 9m                   1315 non-null float64
Recoveries @ 12m                  909 non-null float64
Total Recovery                    3168 non-null float64
Prepayment Amount                 72167 non-null float64
dtypes: float64(11), int64(3), object(6)
memory usage: 11.0+ MB
```

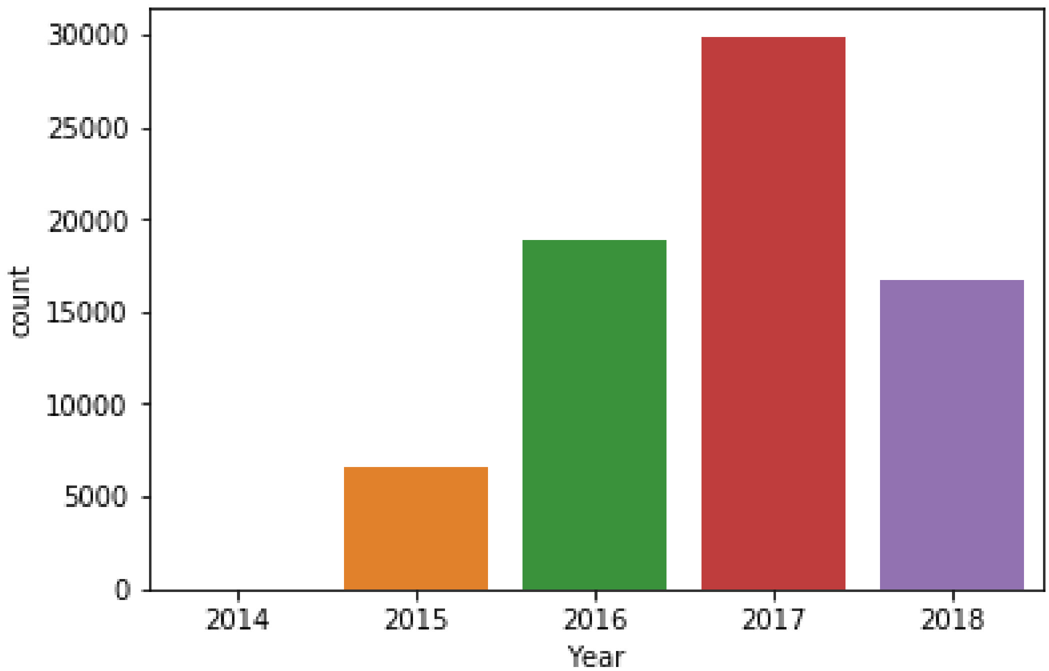
```
The number of unique entries for LoanID in the StaticData DataFrame is 72167
```

```
The number of unique entries for LoanID in the Performance DataFrame is 72167
```

Here's the distribution of loans accross years

```
In [7]: sns.countplot(x="Year",data=StaticData)
```

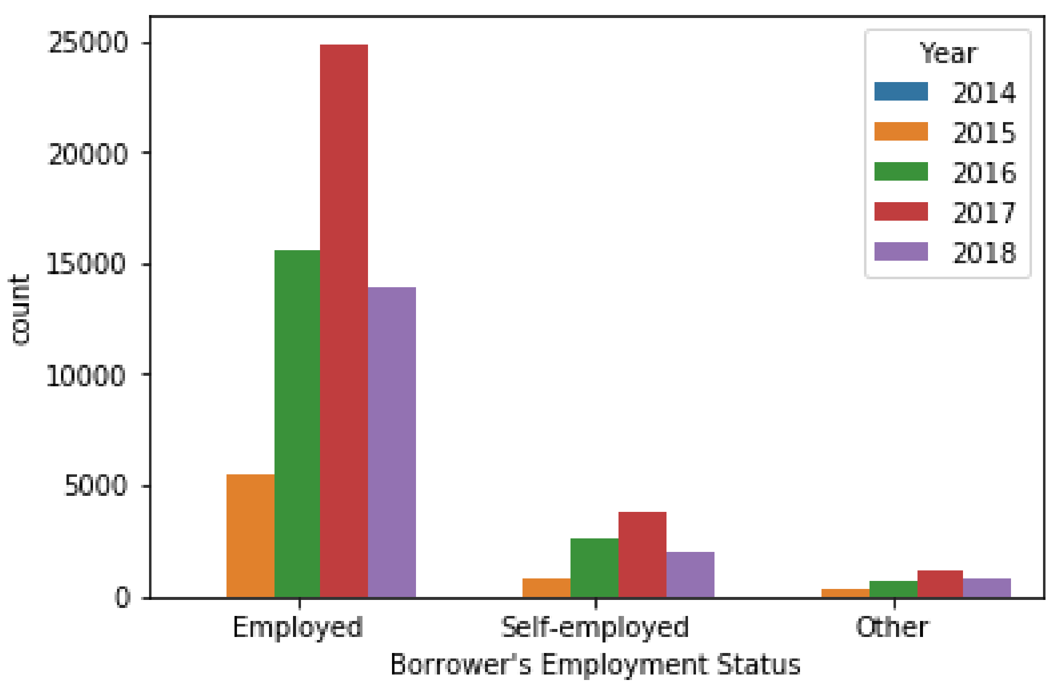
```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x21315ce5a58>
```



It seems that most loans are dated in 2017. Additionally, *Employed* consumers take/are given loans the most regardless of the year.

```
In [8]: sns.countplot(x="Borrower's Employment Status",hue='Year',data=StaticData)
```

```
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x21315cd9f98>
```



We merge the Performance DataFrame with the StaticData DataFrame to obtain performance information for each loan

```
In [9]: # ACTION: Merge
        MergedData = StaticData.merge(Performance.iloc[:,0:3],on='LoanID')
```

What does default within the first 24 months actually mean? We will have to give a specific definition and the definition we give here is 'All cases with default within the first 12 months is a subset of all cases with default within the first 24 months'. We can see below that there is some ambiguity in the *Status 24* column. Specifically, there are 709 cases where there has been a default within the first 12 months but we have 'NA' in the *Status 24* column.

```
In [10]: len(MergedData[(MergedData['Status 12'] == 1) & (MergedData['Status 24'] != 1)])

Out[10]: 709
```

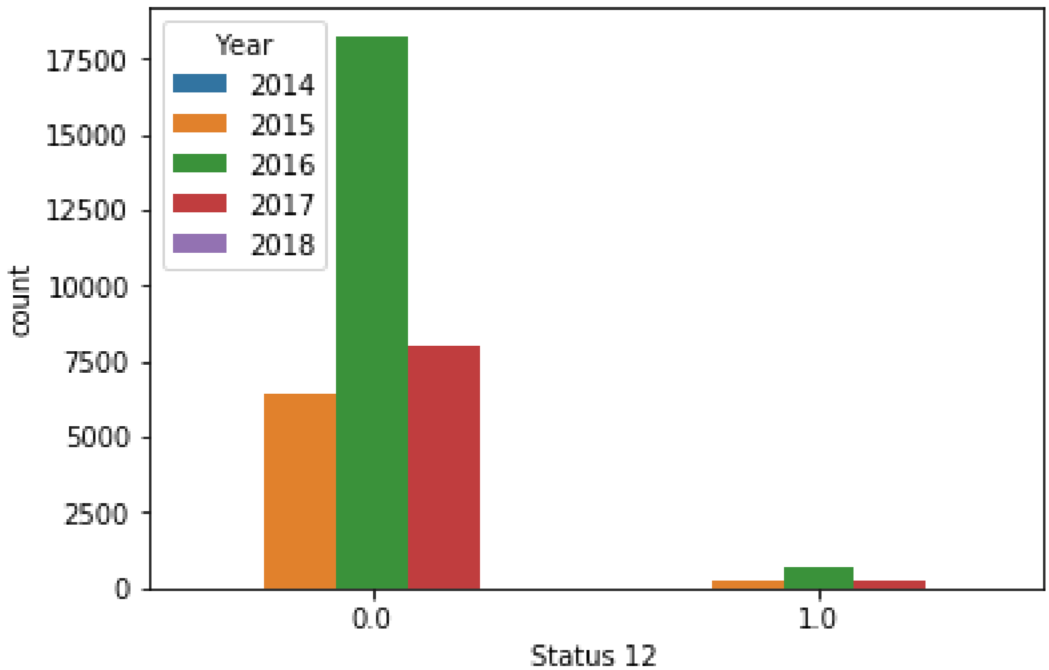
We therefore create a new column called *Default 24* which caters for this. Below we see that there are no longer any cases that allow for a default within the first 12 months but 'NA' in the first 24 months. We also make sure that in cases where there is no default within 12 months, and the *Status 24* column has 'NA' that there is no default within 24 months also.

```
In [11]: # ACTION: Create new Default24 column and use this instead in the models


         def default24(x):
             '''
             This function is used in the creation of the Default24 column in accordance with
             x := A row of a DataFrame with x[0], x[1] the 'Status 12' and 'Status 24' columns
             The logic is: If Status 12 = 1, then Default24 = 1. If Status 12 = 0 and Status 2
             otherwise, Default24 = Status 24.
             '''
             if x[0] == 1:
                 return 1
             elif x[0] == 0 and np.isnan(x[1]):
                 return 0
             else:
                 return x[1]


In [12]: # Create a new column dependent on whether there has been a default within the first
         MergedData['Default24'] = MergedData[['Status 12','Status 24']].apply(lambda x: defaul
         print('There are {} cases of ambiguity with the \'Default24\' column'.format(len(Merge

There are 0 cases of ambiguity with the 'Default24' column
```

We can see that in proportion to the total number of loans, there are very few that have defaulted:

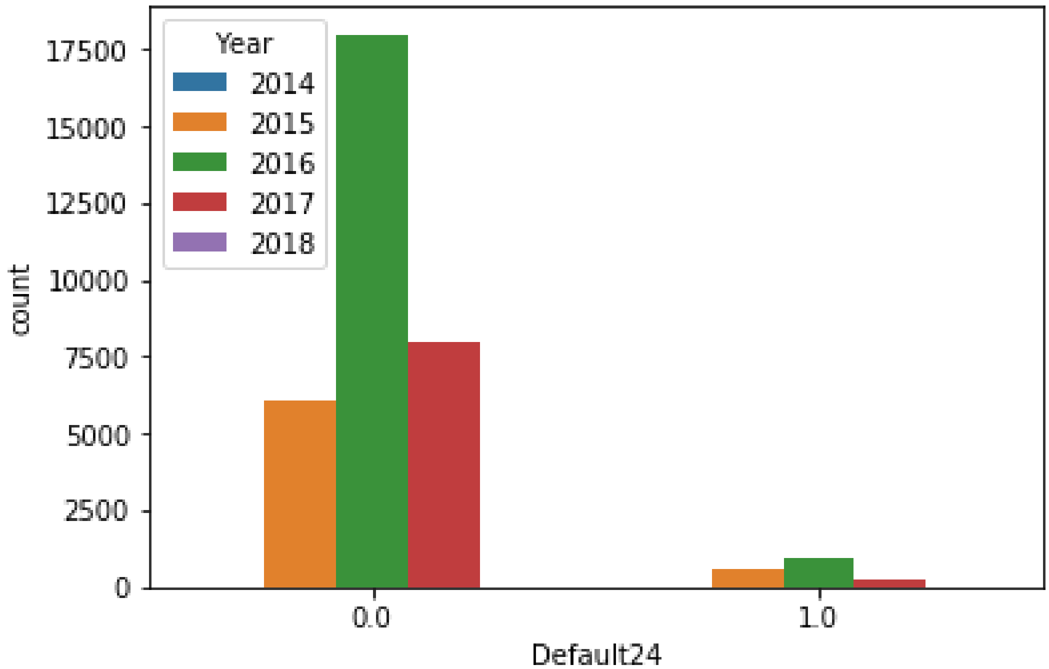
```
In [13]: sns.countplot(x="Status 12",hue='Year',data=MergedData)
```

```
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x213162e5780>
```



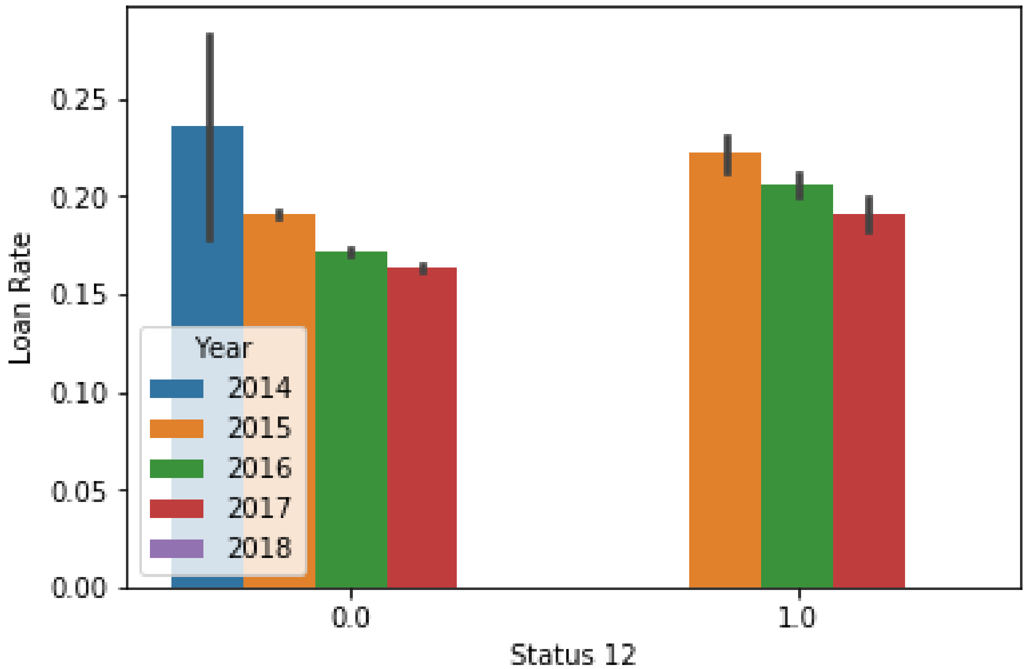

```
In [14]: sns.countplot(x="Default24",hue='Year',data=MergedData)
```

```
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x213162e5630>
```



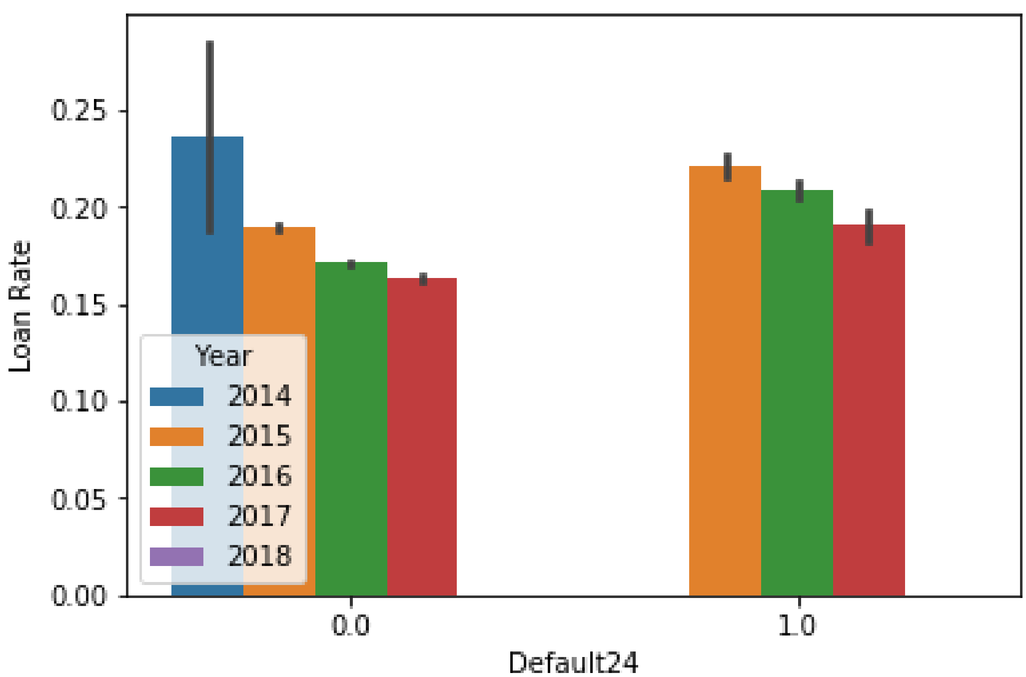
Default Probability It seems that except for year 2014 for which there was no default information, the average *Loan Rate* for those that didn't default is slightly lower than for those that did default. This holds true for both the 12 month and 24 month periods. This suggests that the *Loan Rate* can give us some information as to the probability of default:

```
In [15]: # ACTION: Add loan rate to the model parameters
         sns.barplot(x="Status 12",y='Loan Rate',hue='Year',data=MergedData)

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x21316616710>
```



In [16]: `sns.barplot(x="Default24",y='Loan Rate',hue='Year',data=MergedData)`

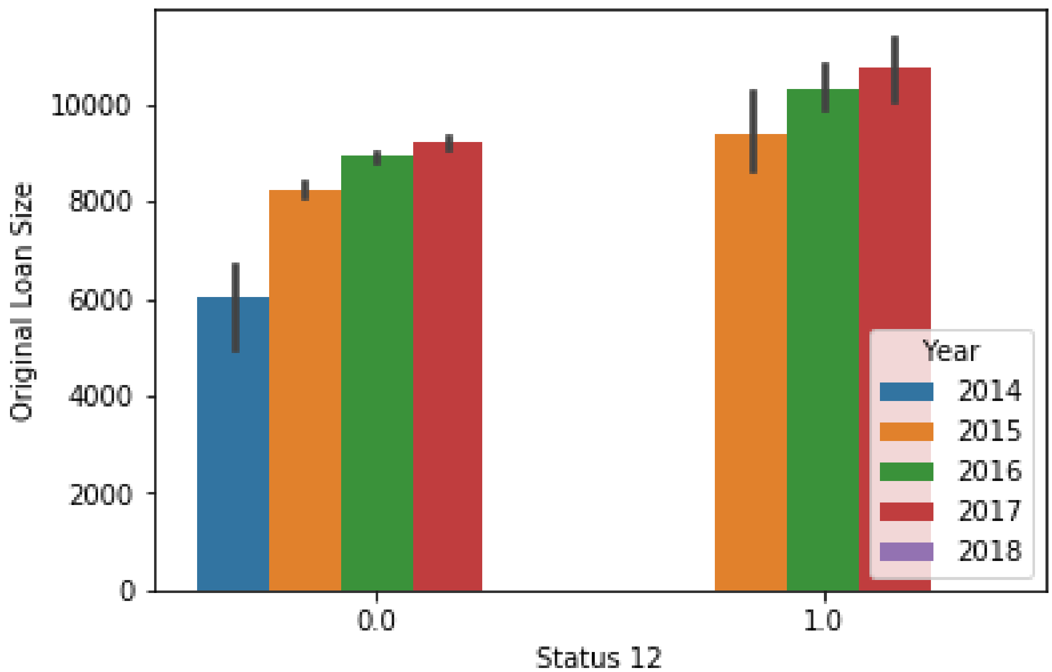
Out[16]: `<matplotlib.axes._subplots.AxesSubplot at 0x21316623390>`



Consumers that default have a consistently larger Original Loan Size on average each year. This may also be an indicator of default:

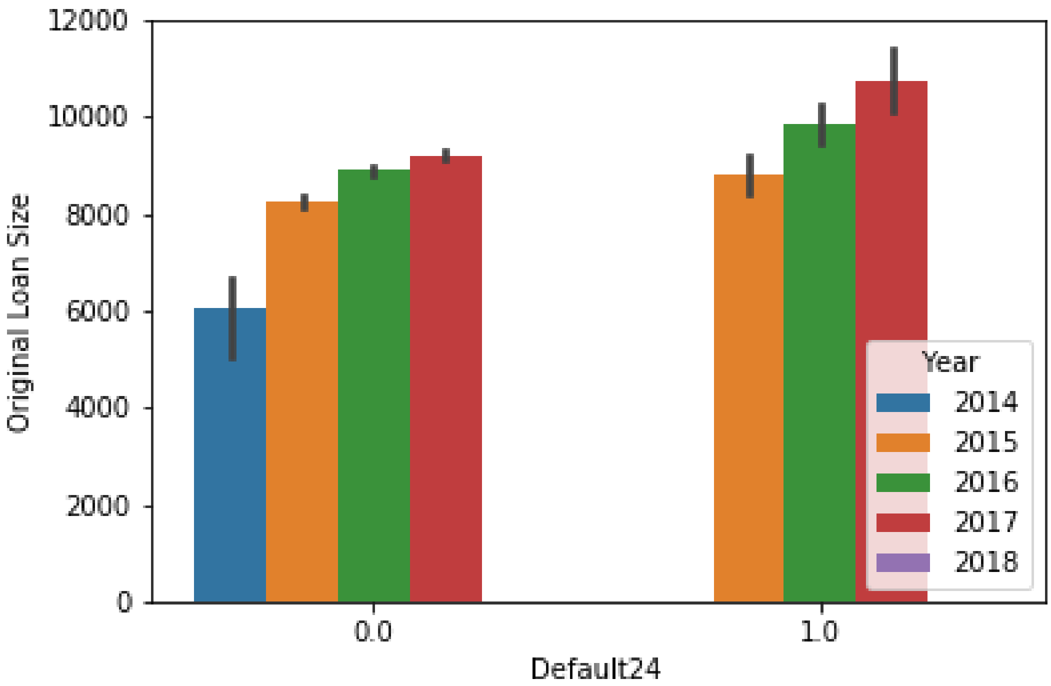
```
In [17]: # ACTION: Add Original Loan Size to the model parameters
         sns.barplot(x="Status 12",y='Original Loan Size',hue='Year',data=MergedData)
```

```
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x213166dee80>
```



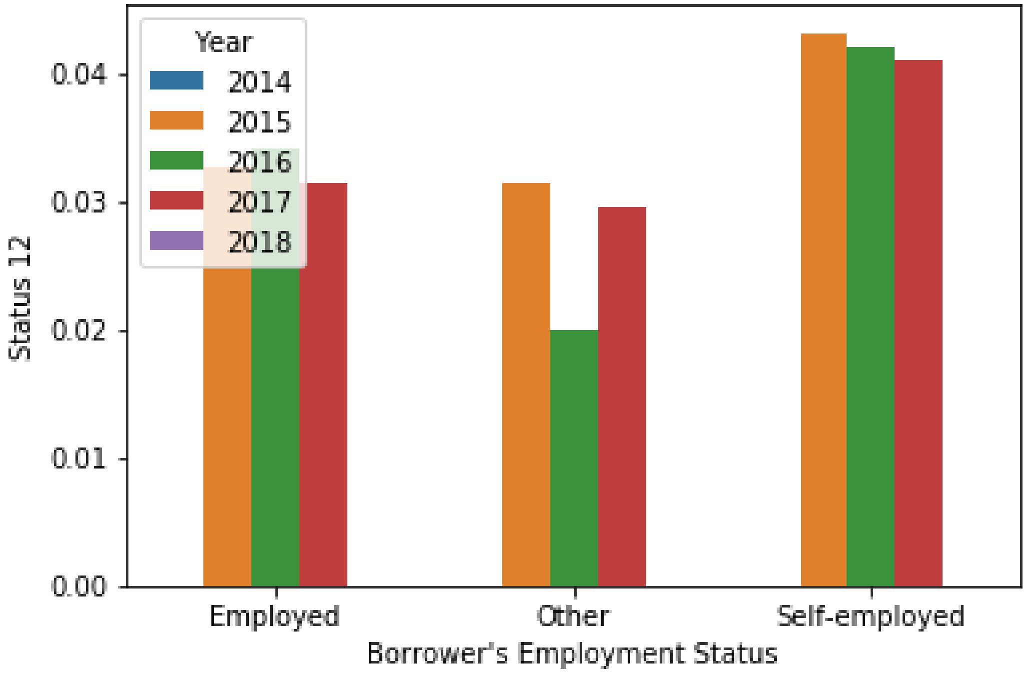

```
In [18]: sns.barplot(x="Default24",y='Original Loan Size',hue='Year',data=MergedData)
```

```
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x21316796f98>
```



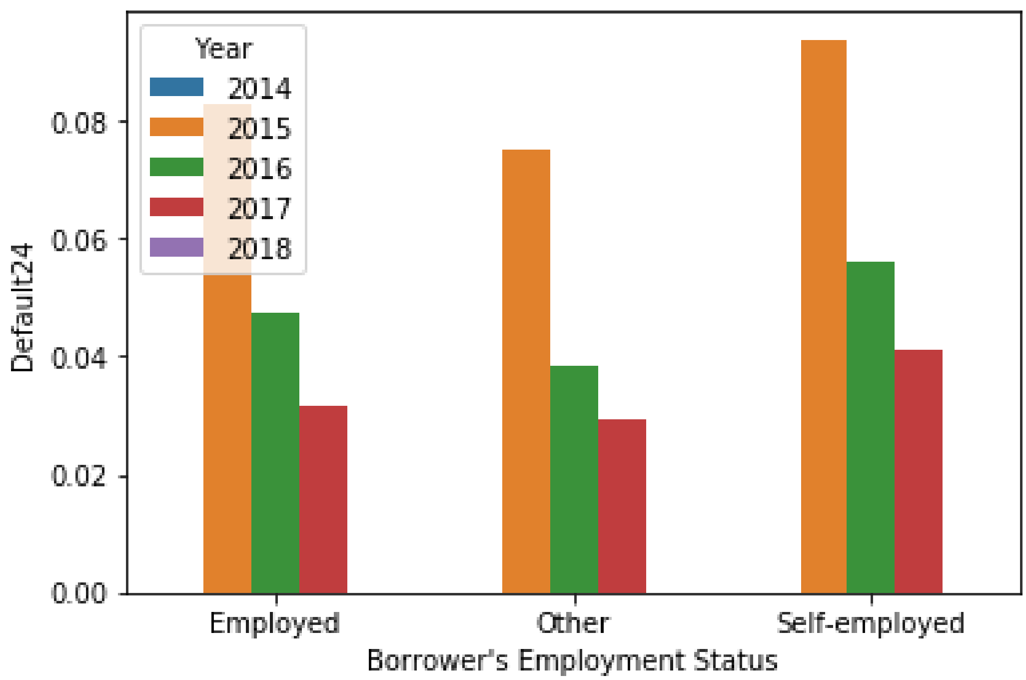
Let's see if *Employment Status* is any indication for default. We can't just do a count plot since we know that more Employed Consumers take out loans. So we find the ratio of default in each category by grouping on *Borrower's Employment Status*. We see that whether a consumer is *Self-employed*, *Employed* or *Other* has an effect on the default probability:

```
In [19]: # ACTION: Create dummy data for Self-employed and other and add them to the model para
         MeanDefault = MergedData[['Year', 'Status 12', 'Status 24','Default24', "Borrower's Em
         sns.barplot(x="Borrower's Employment Status", y="Status 12",hue="Year",data=MeanDefaul

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x2131686a358>
```




```
In [20]: sns.barplot(x="Borrower's Employment Status", y="Default24",hue="Year",data=MeanDefaul
```

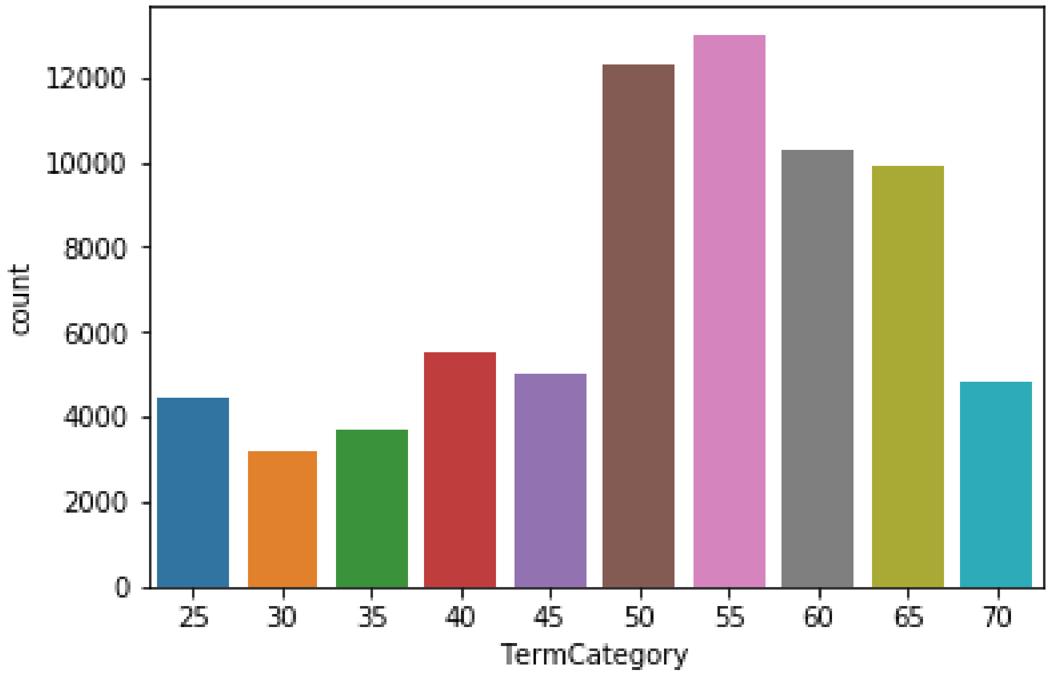
```
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x2131684e5f8>
```

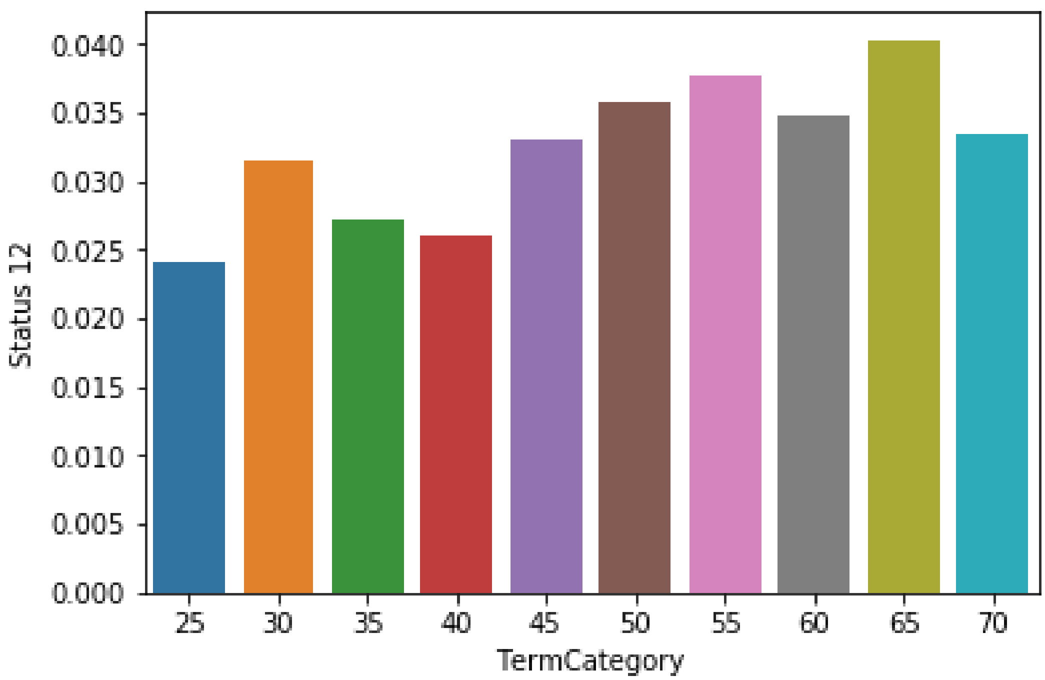


We can see below that as the *Original Term* increases, the proportion of default also increases (although this is less pronounced for the 'Default24' category). We can create bins of 5 and categorise in order to not have too few records in a bucket. We first create a function to categorise Original Term <= 25 as 25 and Original Term >= 70 as 70.

```
In [21]: def getTermCategory(x):
             '''
             A function to be used to even out the number of observation within a term bucket
             within each category.
             x := An element from the 'Original Term' column of the StaticData table.
             Logic: Bucket Original Term <= 25 and >= 70 into their own categories.
             '''
             if x <= 25:
                 return 25
             elif x >= 70:
                 return 70
             else:
                 return x
```

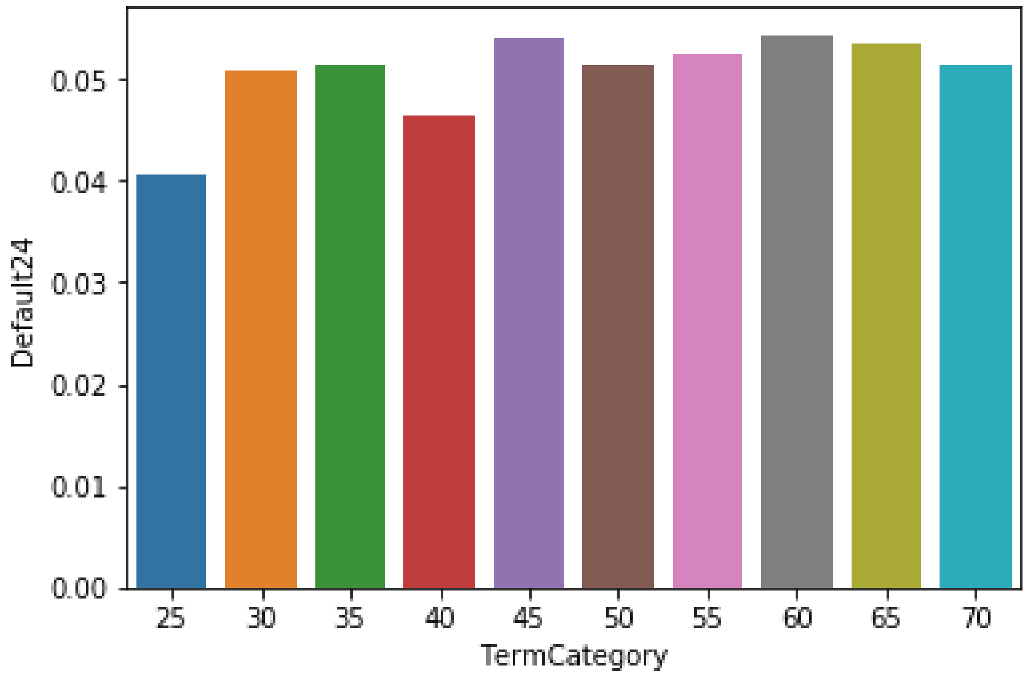
```
In [22]: # ACTION: Create Original Term buckets of 5. Group up >= 75 and <=15 into their own c
         # Create dummy variables. Then add to the model as a parameter.

         # As the term increases, there is a slightly higher chance of default for both 12 and
         # But this is only true for records with Original Term between 15 and 75 since we don

         # We will create buckets of 5
         x = MergedData['Original Term'].apply(lambda x: int(x/5)*5)

         xdf = pd.DataFrame(x)
         xdf['TermCategory'] = xdf['Original Term'].apply(lambda x: getTermCategory(x))

         # There are only a small amount of non null records with original term > 70 and < 15
         sns.countplot(xdf['TermCategory'])
         plt.show()
         sns.barplot(x=xdf['TermCategory'], y=MergedData["Status 12"],ci=0)
         plt.show()
```






```
In [23]: sns.barplot(x=xdf['TermCategory'], y=MergedData["Default24"],ci=0)
```

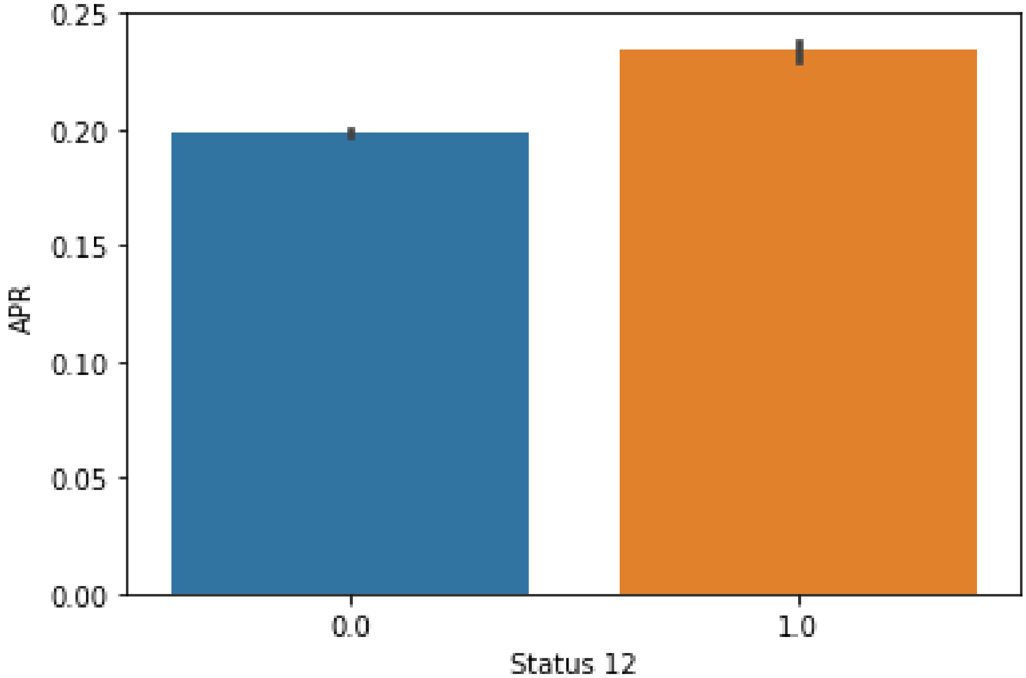
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x21316a5eeb8>



APR is on average higher for those that default.

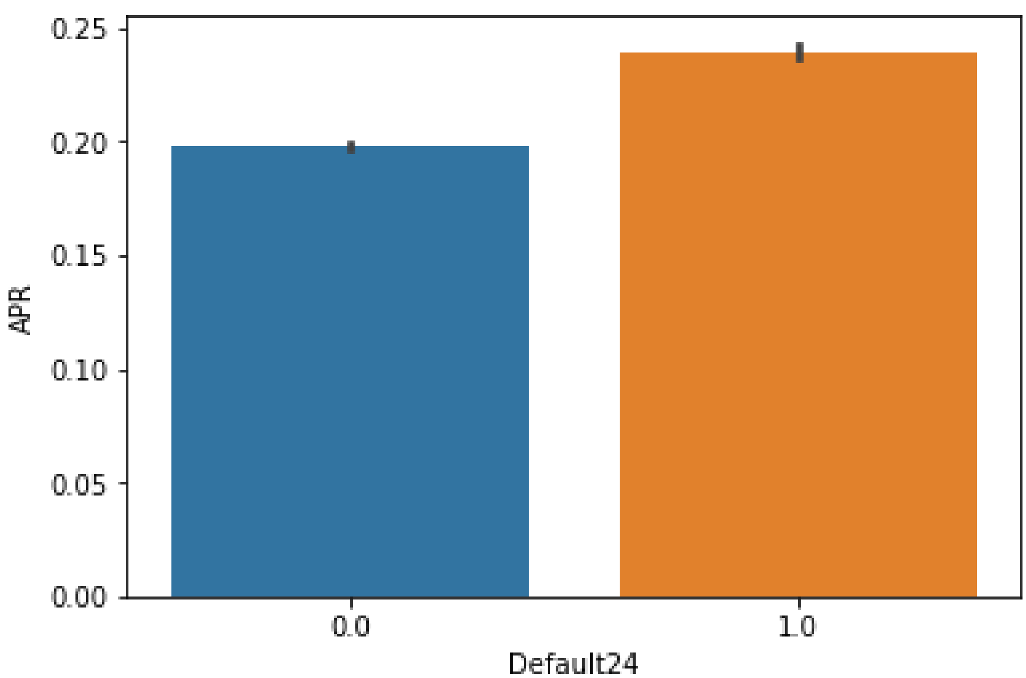
In [24]:
```python
# ACTION: Add APR as a continuous variable to the model

# Those that default have higher APR on average
sns.barplot(x="Status 12", y="APR",data=MergedData)
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x21316b4a0b8>




```
In [25]: # Those that default have higher APR on average
         sns.barplot(x="Default24", y="APR",data=MergedData)
```

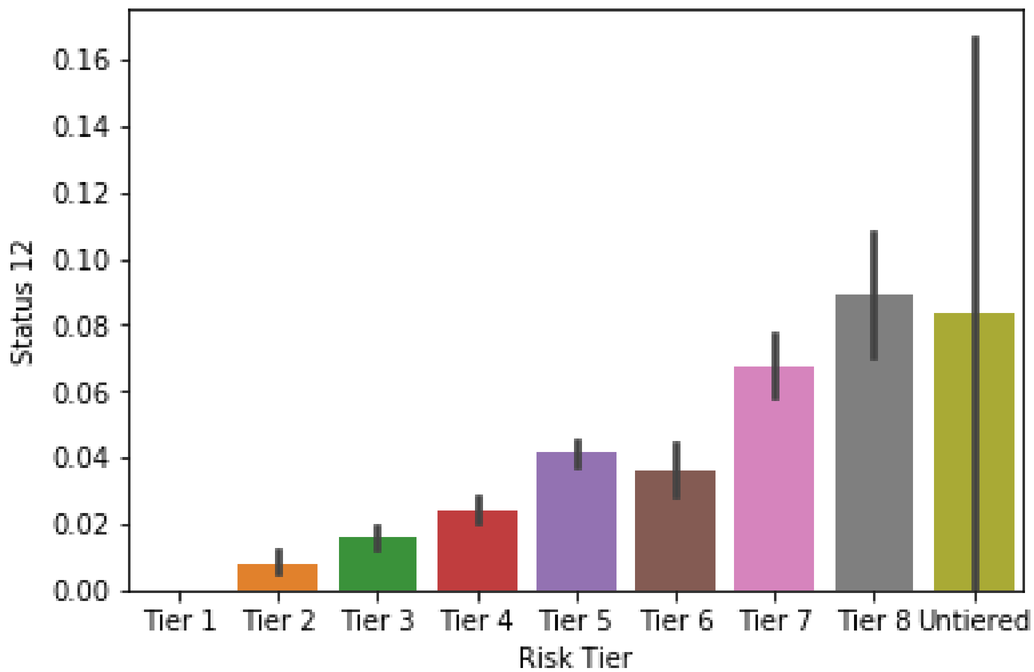
```
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x2131ae28cf8>
```



Tier is a good indicator since higher tiers correspond to higher proportion of default for both the 12 month and 24 month periods.

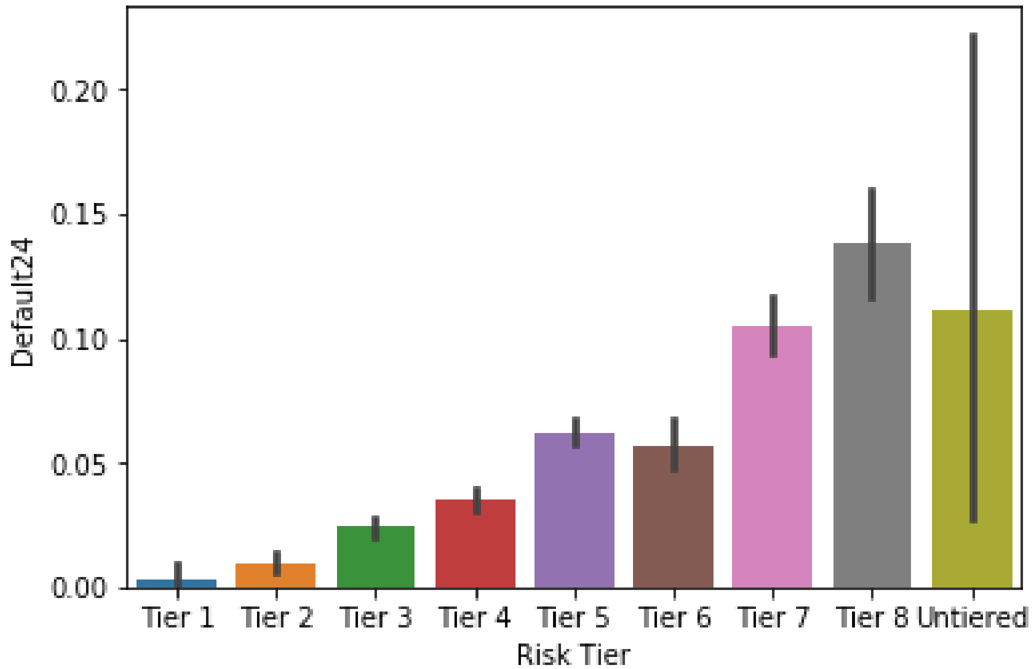
```
In [26]: # ACTION: Create dummy indicators for Tier and add it to the model.

         # Tier looks to be an indicator
         sns.barplot(x="Risk Tier", y="Status 12",data=MergedData,order=['Tier 1','Tier 2','Tie
```

```
Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x213169e59e8>
```




```
In [27]: # Tier may be an indicator
         sns.barplot(x="Risk Tier", y="Default24",data=MergedData,order=['Tier 1','Tier 2','Tie
```

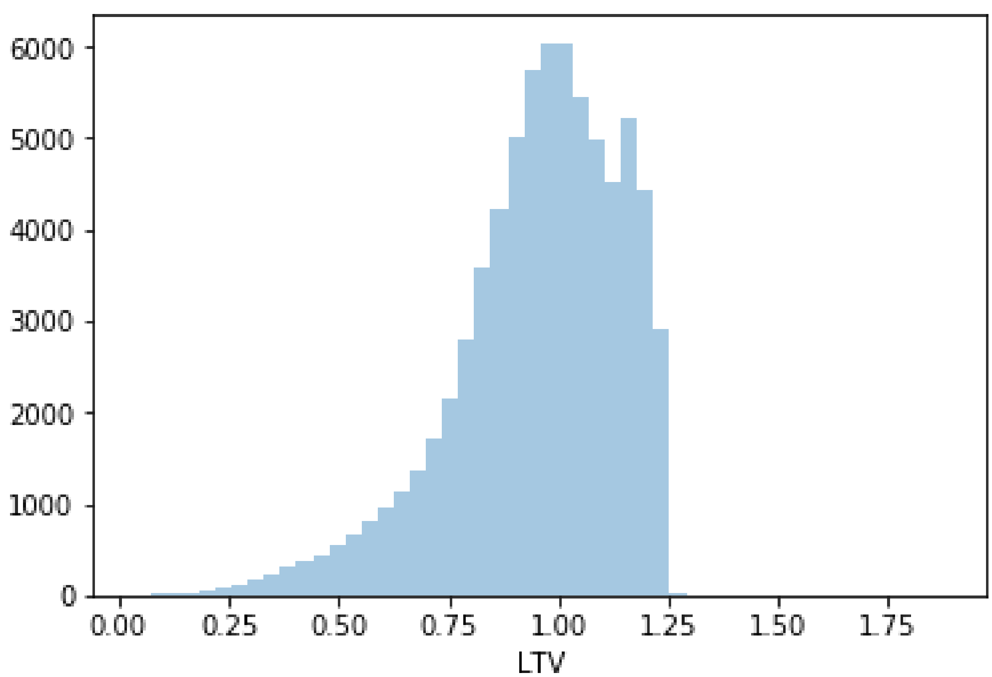
```
Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x2131ae45358>
```



LTV seems to be correlated with default proportion. The distribution of LTV is right skewed.

```
In [28]: sns.distplot(MergedData["LTV"],kde=False)
```

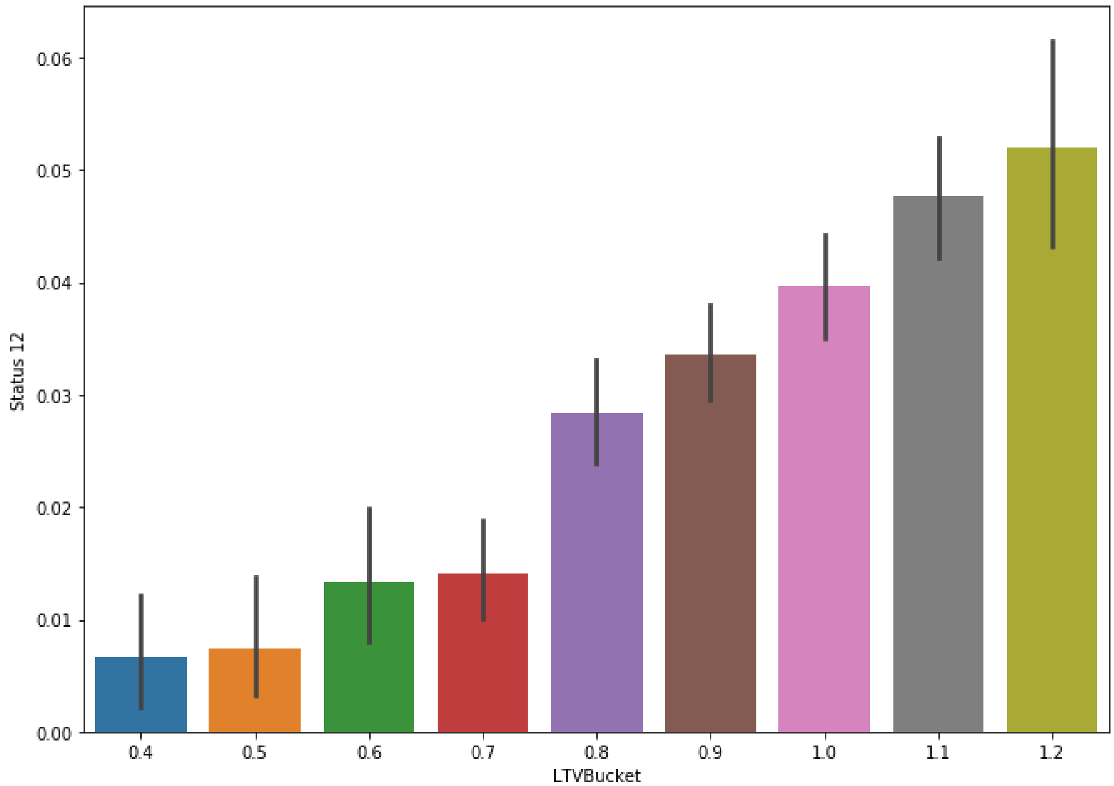
```
Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x2131aed0cc0>
```



In [29]:
```
def getLTVCategory(x):
    '''
    A function to be used to bucket the LTV to even out observation within each bucke
    x := An element in from the LTV column of the StaticData DataFrame.
    Logic: Bucket LTV <= 0.4 and >= 1.2 into their own buckets.
    '''
    if x <= 0.4:
        return 0.4
    elif x >= 1.2:
        return 1.2
    else:
        return round(int(x/0.1)*0.1,2)
```

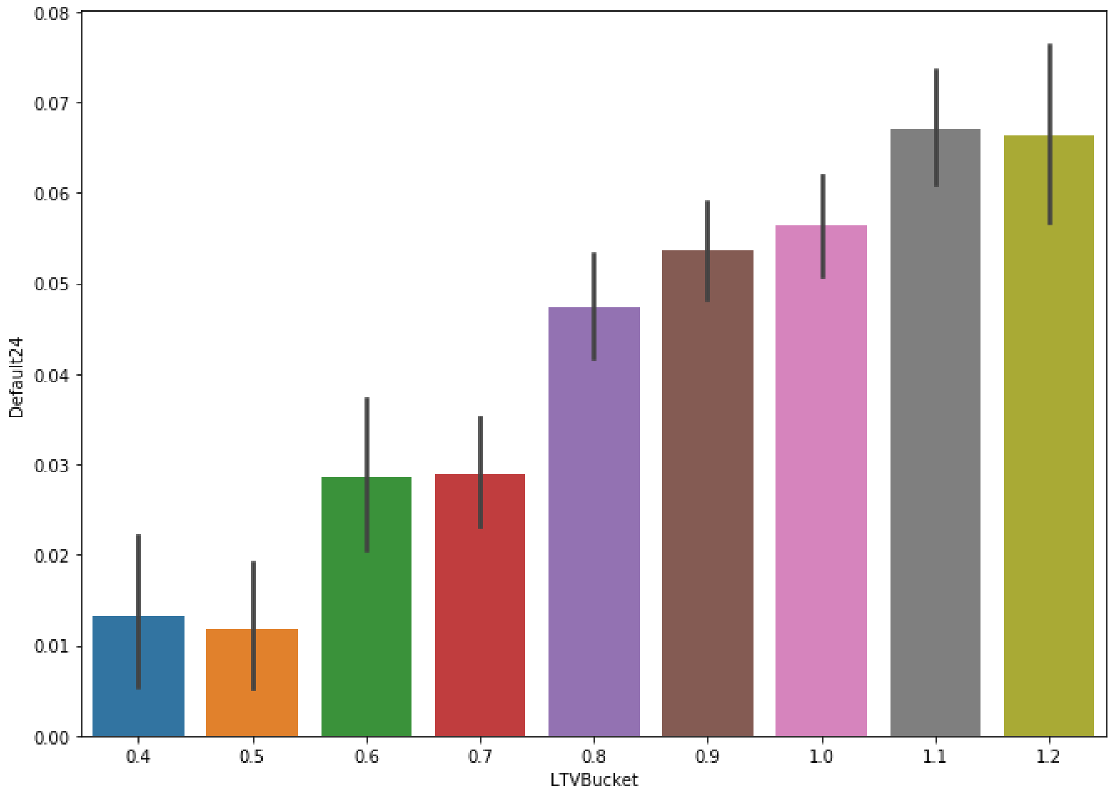
In [30]:
```
# ACTION: Create LTV buckets. Add this as a numerical feature to the model parameters

MergedData['LTVBucket'] = MergedData['LTV'].apply(getLTVCategory)

fig, ax = plt.subplots()
# the size of A4 paper
fig.set_size_inches(11, 8)

sns.barplot(x="LTVBucket", y="Status 12",data=MergedData)
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x2131afa8e48>



In [31]:
```
fig, ax = plt.subplots()
# the size of A4 paper
fig.set_size_inches(11, 8)

sns.barplot(x="LTVBucket", y="Default24",data=MergedData)
```

Out[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2131afbaf28>
```

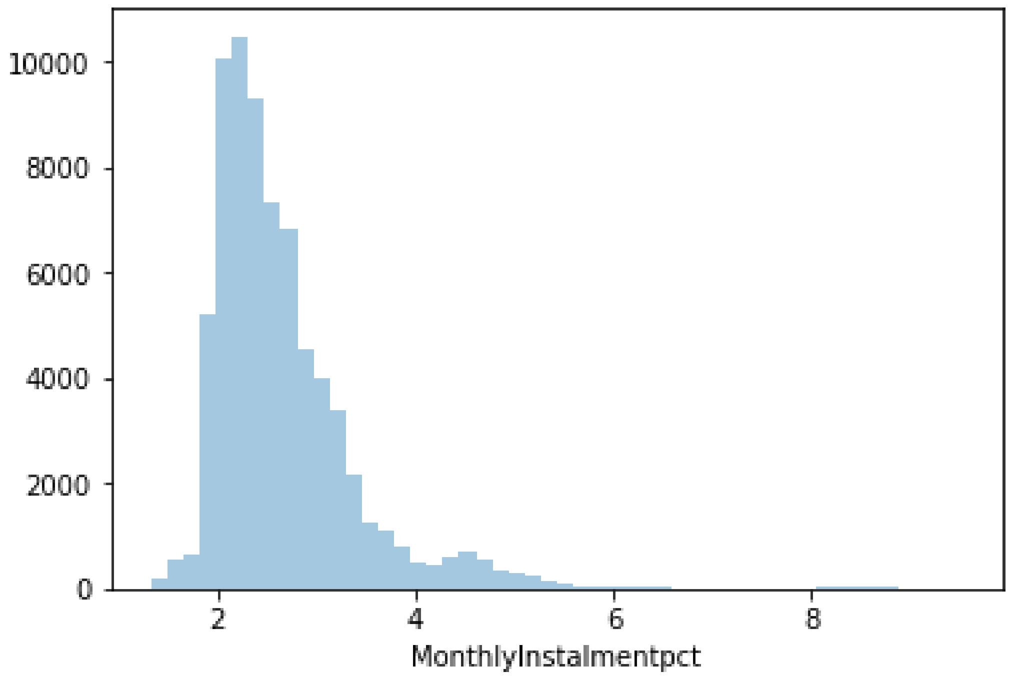


The *Monthly Instalment* as a percentage of the *Original Loan Size* looks to be related to an increased proportion of defaults.

```
In [32]: # ACTION: Create a Monthly Instalment pct column. Bucket these. Then add to the model

         MergedData['MonthlyInstalmentpct'] = MergedData[['Original Loan Size','Monthly Instalm
```

```
In [33]: sns.distplot(MergedData["MonthlyInstalmentpct"],kde=False)
```

```
Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x2131b00d908>
```

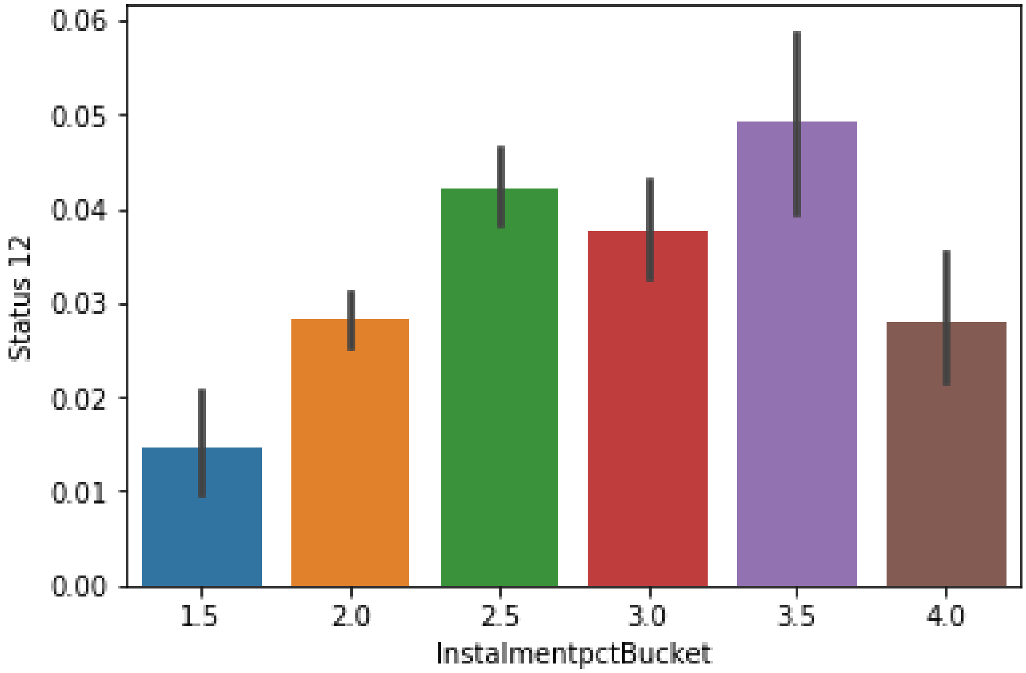


In [34]:
```
def getInstalmentpctCategory(x):
    '''
    A function to be used to bucket the 'Monthly Instalment' to even out observation i
    bucket so a default proportion can be observed.
    x := An element in from the 'Monthly Instalment' column of the StaticData DataFra
    Logic: Bucket Monthly Instalment <= 1.5 and >= 4 into their own buckets.
    '''
    if x <= 1.5:
        return 1.5
    elif x >= 4:
        return 4
    else:
        return int(x/0.5)*0.5
```

In [35]:
```
MergedData['InstalmentpctBucket'] = MergedData['MonthlyInstalmentpct'].apply(getInstal
```

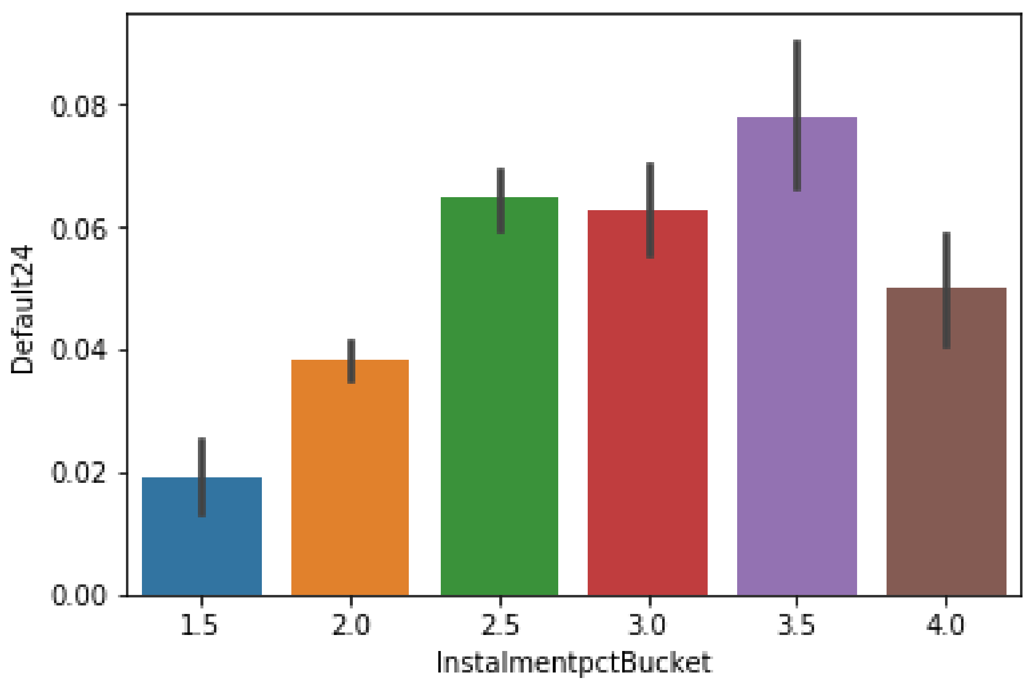
In [36]:
```
sns.barplot(x="InstalmentpctBucket", y="Status 12",data=MergedData)
```

Out[36]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2131b0f7ba8>
```



In [37]: `sns.barplot(x="InstalmentpctBucket", y="Default24",data=MergedData)`

Out[37]: `<matplotlib.axes._subplots.AxesSubplot at 0x2131c1ae908>`



```
In [38]: statusNull122015 = pd.isna(MergedData[(MergedData['Year'] == 2015)]['Status 12']).sum
         statusNull242015 = pd.isna(MergedData[(MergedData['Year'] == 2015)]['Status 24']).sum

         print('{} % of the data in 2015 have non-null \'Status 12\' values'.format((1-statusNu
         print('{} % of the data in 2015 have non-null \'Status 24\' values\n'.format((1-statu



         statusNull122016 = pd.isna(MergedData[(MergedData['Year'] == 2016)]['Status 12']).sum
         statusNull242016 = pd.isna(MergedData[(MergedData['Year'] == 2016)]['Status 24']).sum

         print('{} % of the data in 2016 have non-null \'Status 12\' values'.format((1-statusNu
         print('{} % of the data in 2016 have non-null \'Status 24\' values\n'.format((1-statu



         statusNull122017 = pd.isna(MergedData[(MergedData['Year'] == 2017)]['Status 12']).sum
         statusNull242017 = pd.isna(MergedData[(MergedData['Year'] == 2017)]['Status 24']).sum

         print('{} % of the data in 2017 have non-null \'Status 12\' values'.format((1-statusNu
         print('{} % of the data in 2017 have non-null \'Status 24\' values\n'.format((1-statu



         statusNull122018 = pd.isna(MergedData[(MergedData['Year'] == 2018)]['Status 12']).sum
         statusNull242018 = pd.isna(MergedData[(MergedData['Year'] == 2018)]['Status 24']).sum

         print('{} % of the data in 2018 have non-null \'Status 12\' values'.format((1-statusNu
         print('{} % of the data in 2018 have non-null \'Status 24\' values'.format((1-statusNu
```

```
100.0 % of the data in 2015 have non-null 'Status 12' values
100.0 % of the data in 2015 have non-null 'Status 24' values

100.0 % of the data in 2016 have non-null 'Status 12' values
29.11774039987306 % of the data in 2016 have non-null 'Status 24' values

27.69503664781284 % of the data in 2017 have non-null 'Status 12' values
0.0 % of the data in 2017 have non-null 'Status 24' values

0.0 % of the data in 2018 have non-null 'Status 12' values
0.0 % of the data in 2018 have non-null 'Status 24' values
```

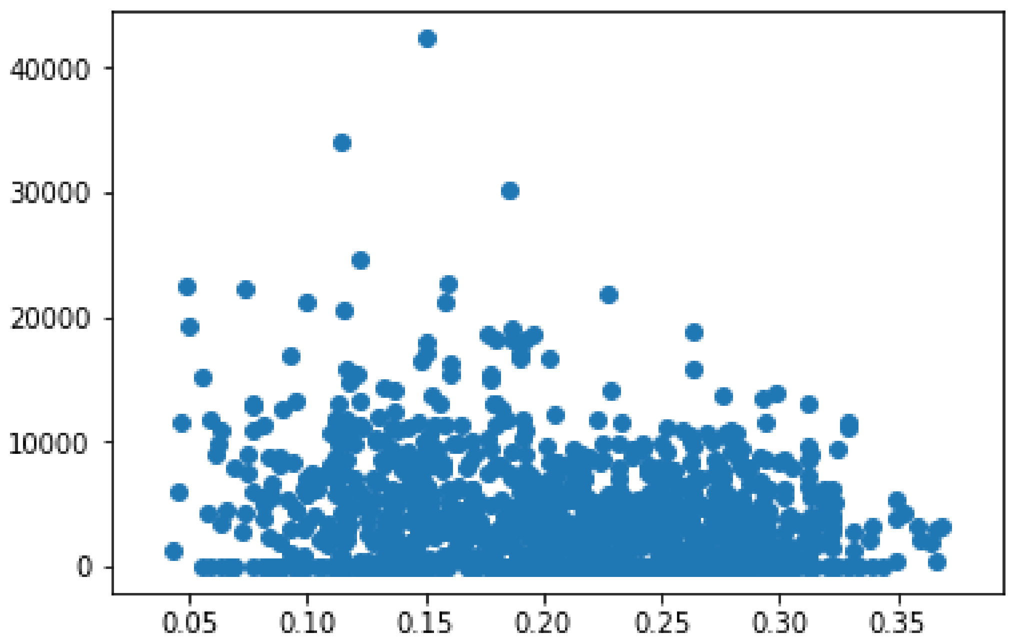
Modelling Value as a function of Loan Rate We may be able to model the **Value=Original Loan Size** × **Total Recovery** as a function of *Loan Rate* and *Original Term*.

Here we are only interested in the consumers that have a non-null *Exposure At Default* because we want to obtain a model for those that have defaulted. We see below that *Value* looks to be a non-linear function of *Loan Rate* and *Total Recovery*. We get a much more linear scatter plot when we use **Loan Rate**^{2}.

In [39]:
```
# ACTION: Create a Value column. Add the parameters Loan Rate and Original Term to a

MergedData['Value'] = MergedData['Original Loan Size']*MergedData['Total Recovery']
MergedDataDefaulted12 = MergedData[(MergedData['Status 12'] == 1) & (pd.isna(MergedDa
MergedDataDefaulted24 = MergedData[(MergedData['Default24'] == 1) & (pd.isna(MergedDa

# MergedDataDefaulted12 = MergedData[MergedData['Status 12'] == 1]
# MergedDataDefaulted24 = MergedData[MergedData['Status 24'] == 1]
```

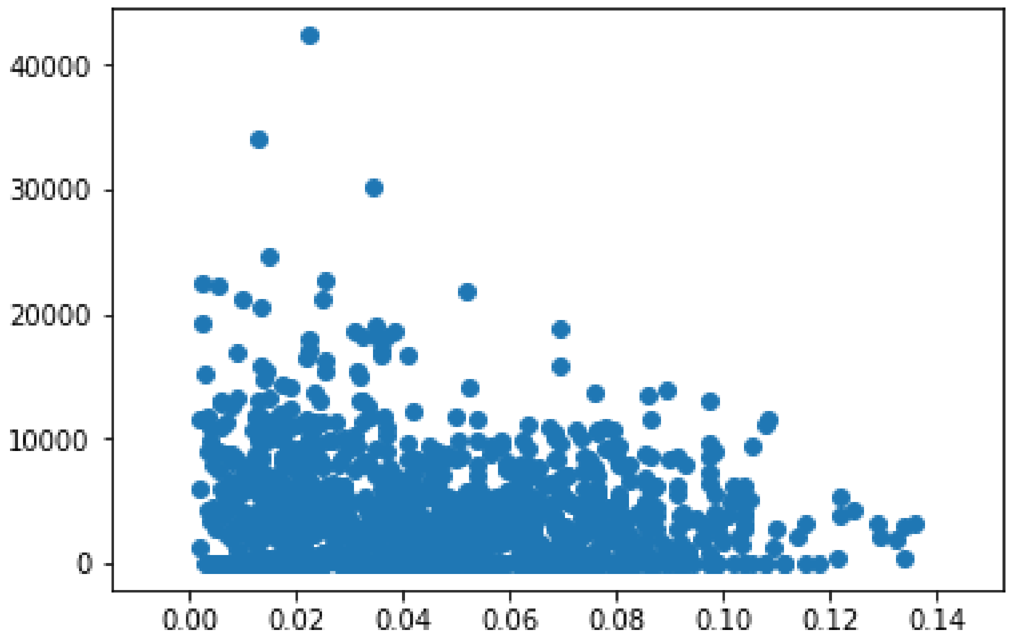
In [41]:
```
# Let's obtain a relationship with Loan Rate
plt.scatter(x=((MergedDataDefaulted12['Loan Rate'])),y=MergedDataDefaulted12['Value']]
#plt.ylim(0,0.05)
```

Out[41]: <matplotlib.collections.PathCollection at 0x2131c31bcc0>

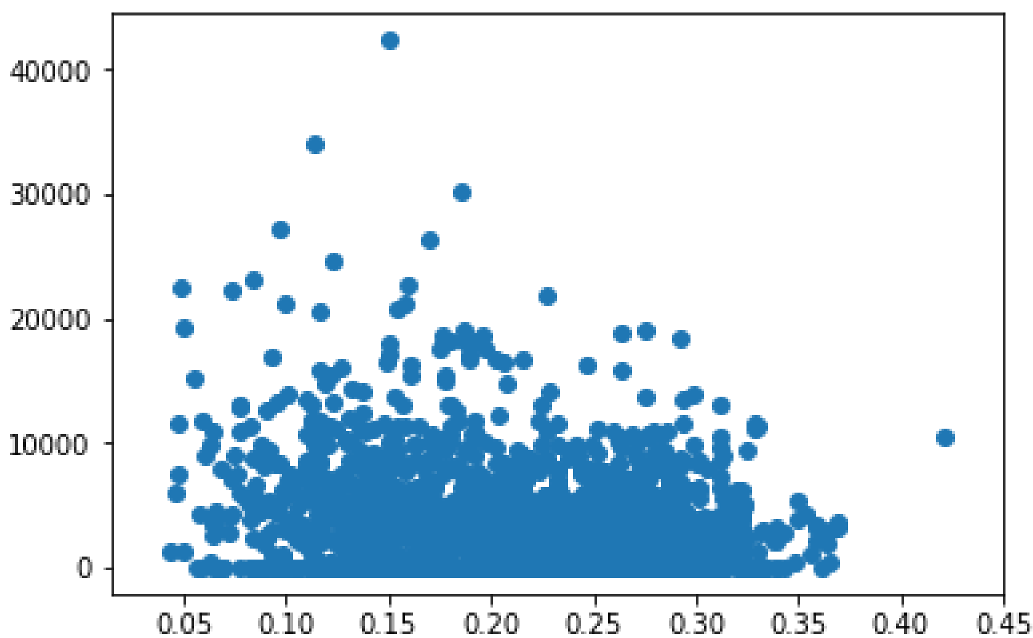


In [129]:
```
plt.scatter(x=((MergedDataDefaulted12['Loan Rate']**2)),y=MergedDataDefaulted12['Valu
```

Out[129]: <matplotlib.collections.PathCollection at 0x213232c51d0>

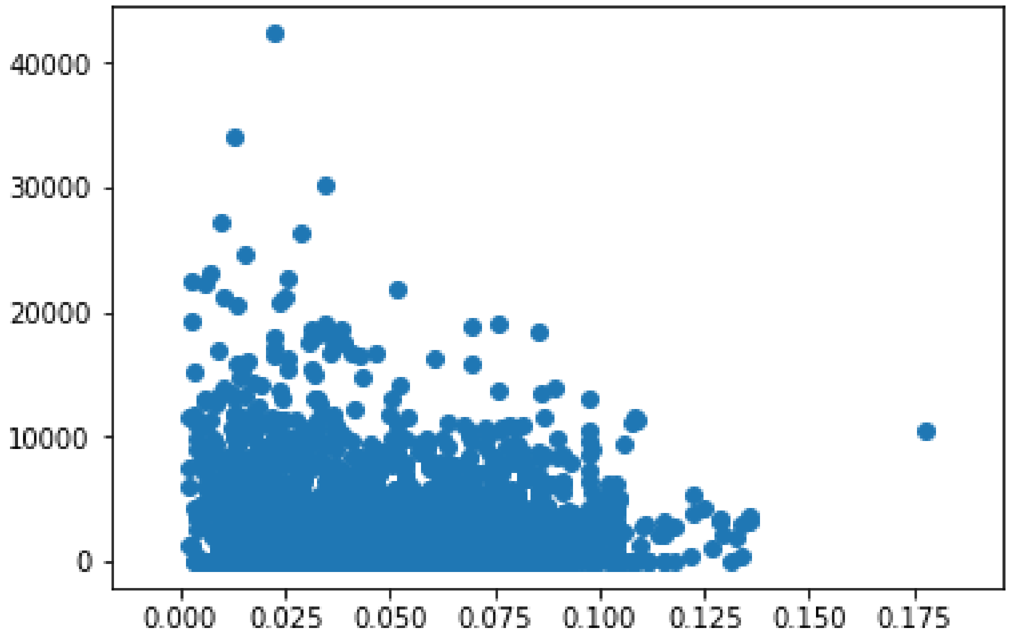



```
In [130]: # Let's obtain a relationship with APR
          plt.scatter(x=((MergedDataDefaulted24['Loan Rate'])),y=MergedDataDefaulted24['Value']
          #plt.ylim(0,2)

Out[130]: <matplotlib.collections.PathCollection at 0x213235a8898>
```



```
In [131]: plt.scatter(x=((MergedDataDefaulted24['Loan Rate']**2)),y=MergedDataDefaulted24['Valu
```

```
Out[131]: <matplotlib.collections.PathCollection at 0x213237a04e0>
```



Transformation and Cleaning Our transform and clean function should have the following actions from our analysis above:

0- ACTION: Merge
1- ACTION: Create new Default24 column and use this instead in the models
2- ACTION: Create dummy data for Self-employed
3- ACTION: Create Original Term buckets of 5. Group up >= 75 and <=15 into their own categories
4- ACTION: Create dummy indicators for Tier
5- ACTION: Create LTV buckets
6- ACTION: Create a Monthly Instalment pct column and bucket them
7- ACTION: Create an Value column
8- ACTION: Create Loan Rateˆ2 and Loan Rate*OriginalTerm Columns

The Model should have the following parameters:

1- ACTION: Add loan rate to the model parameters
2- ACTION: Add Original Loan Size to the model parameters
3- ACTION: Add dummy data for Employment Status to the model parameters
4- ACTION: Add Term Buckets to model parameters
5- ACTION: Add APR as a continuous variable to the model

6- ACTION: Add dummy data for Term to the model
7- ACTION: Add LTV buckets as a numerical feature to the model parameters
8- ACTION: Add the bucketed Monthly Instalment pct buckets to the model as a parameter
9- ACTION: Add the parameters Loan Rate^2 and Original Term to a Linear Regression model with both mixed effect (synergy effect) and separately in order to model ExposureRatio

```
In [48]: def TransformMergeAndCleanData(df,df2):
             '''
             A function used to merge, transform and clean the dataframes according to the abo
             df := The StaticData dataframe
             df2 := The Performance dataframe
             '''
             #0- Merge the two DataFrames
             df_out = df.merge(df2.iloc[:,0:3],on='LoanID')

             #1- Create new Default24 column and use this instead in the models
             df_out['Default24'] = df_out[['Status 12','Status 24']].apply(lambda x: default24

             # Convert the Origination Date to date object
             df_out['OriginationDateFormatted'] = df_out['Origination Date'].apply(lambda x: da

             #2- Employment Status is either Employed, Unemployed or other. Create a series wh
             Employment = pd.get_dummies(df_out['Borrower\'s Employment Status'],drop_first=Tru

             #3- We will create buckets of 5 for Original Term
             x = df_out['Original Term'].apply(lambda x: int(x/5)*5)
             xdf = pd.DataFrame(x)
             OriginalTermBucket = xdf['Original Term'].apply(lambda x: getTermCategory(x))
             OriginalTermBucket.name='OriginalTermBucket'

             #4- Create dummy indicators for Tier
             RiskTier = pd.get_dummies(df_out['Risk Tier'],drop_first=True)

             #5- Create LTV buckets
             LTVBucket = df_out['LTV'].apply(getLTVCategory)
             LTVBucket.name = 'LTVBucket'

             #6- Create a Monthly Instalment pct column and bucket them
             df_out['MonthlyInstalmentpct'] = df_out[['Original Loan Size','Monthly Instalment
             df_out['InstalmentpctBucket'] = df_out['MonthlyInstalmentpct'].apply(getInstalmen

             #7- Create an ExposureRatio column
             df_out['Value'] = df_out[['Original Loan Size','Total Recovery']].apply(lambda x:

             #8- Create APR^2 and APR*OriginalTerm Columns
             df_out['LoanRate2'] = df_out['Loan Rate']**2
             df_out['LoanRate2OriginalTerm'] = df_out['LoanRate2']*df_out['Original Term']
             df_out['LoanRateOriginalTerm'] = df_out['Loan Rate']*df_out['Original Term']
```

```python
    Type1 = pd.get_dummies(df['Type 1'],drop_first=True)

    #Concat the series to the data frame
    df_out = pd.concat([df_out,Employment,OriginalTermBucket,RiskTier,LTVBucket,Type1

    # Remove records which have null Status 12 and Default24 entries separately
    return df_out[pd.isna(df_out['Status 12']).apply(lambda x: not x)], df_out[pd.isna
```

In [49]:
```python
df_final12,df_final24 = TransformMergeAndCleanData(StaticData,Performance)
```

In [50]:
```python
df_final12.iloc[:,20:].head()
```

Out[50]:
```
   Status 12  Status 24  Default24 OriginationDateFormatted  \
1        0.0        0.0        0.0               2014-12-03
2        0.0        0.0        0.0               2014-12-16
3        0.0        0.0        0.0               2014-12-17
4        0.0        0.0        0.0               2014-12-18
5        0.0        0.0        0.0               2014-12-22

   MonthlyInstalmentpct  InstalmentpctBucket  Value  LoanRate2  \
1              3.230143                  3.0    NaN   0.053547
2              2.985235                  2.5    NaN   0.081245
3              3.310211                  3.0    NaN   0.059918
4              3.342916                  3.0    NaN   0.105502
5              2.619870                  2.5    NaN   0.014856

   LoanRate2OriginalTerm  LoanRateOriginalTerm ...  Tier 3  Tier 4  Tier 5  \
1               1.820602              7.867685 ...       0       0       0
2               5.199698             18.242276 ...       0       0       1
3               2.696304             11.015157 ...       0       0       1
4               4.536578             13.966849 ...       0       0       1
5               0.698217              5.728543 ...       0       0       0

   Tier 6  Tier 7  Tier 8  Untiered  LTVBucket  B  C
1       0       0       0         0        0.8  1  0
2       0       0       0         0        0.9  1  0
3       0       0       0         0        0.8  0  0
4       0       0       0         0        0.7  1  0
5       0       0       0         0        0.9  1  0

[5 rows x 24 columns]
```

In [51]:
```python
# What is our baseline model performance metric?
numDefault12 = len(df_final12[df_final12['Status 12'] == 1])
numDefault24 = len(df_final24[df_final24['Default24'] == 1])

# If we classify all consumers to not default...
base12 = (len(df_final12['Status 12']) - numDefault12)/len(df_final12['Status 12'])
```

```
base24 = (len(df_final24['Default24']) - numDefault24)/len(df_final24['Default24'])

print('The base prediction probability for default within 12 months is {}'.format(base
print('The base prediction probability for default within 24 months is {}'.format(base
```

```
The base prediction probability for default within 12 months is 0.9660670966215017
The base prediction probability for default within 24 months is 0.948730844328738
```

1.3 Regression

Logistic Regression We've seen above that the following metrics may be related to an increase in default probability

1- ACTION: Add loan rate to the model parameters
2- ACTION: Add Original Loan Size to the model parameters
3- ACTION: Add dummy data for Employment Status to the model parameters
4- ACTION: Add Term Buckets to model parameters
5- ACTION: Add APR as a continuous variable to the model - REMOVE
6- ACTION: Add dummy data for Risk Tier to the model
7- ACTION: Add LTV buckets as a numerical feature to the model parameters
8- ACTION: Add the bucketed Monthly Instalment pct buckets to the model as a parameter
9- ACTION: Add the parameters Loan Rate^2 and Original Term to a Linear Regression model with both mixed effect (synergy effect) and separately in order to model ExposureRatio

```
In [493]: logisticParams = ['Loan Rate','Original Loan Size',\
                                          'Tier 2','Tier 3','Tier 4','Tier 5', 'Tier 6', 'Tier 7
                                         'Self-employed','Other','OriginalTermBucket','LTVBucket

          linearParams = ['Loan Rate','LoanRate2','LoanRateOriginalTerm','LoanRate2OriginalTer
```

Here we look at the logistic regression performed using the above selected parameters:

```
In [496]: # Select the parameters
          X = df_final12.reset_index()[logisticParams]
          # Select the response variable
          y = df_final12.reset_index()['Status 12']

          # Apply the logistic regression model from statsmodels
          #X = sm.add_constant(X)
          model = sm.Logit(y, X)
          logistic12 = model.fit()
          print(logistic12.summary2())
          print(logistic12.params)
```

```
Optimization terminated successfully.
         Current function value: 0.142770
         Iterations 9
                         Results: Logit
=====================================================================
```

```
Model:                  Logit            No. Iterations:    9.0000
Dependent Variable:     Status 12        Pseudo R-squared:  0.036
Date:                   2018-10-14 17:34 AIC:               9681.8017
No. Observations:       33802            BIC:               9808.2259
Df Model:               14               Log-Likelihood:    -4825.9
Df Residuals:           33787            LL-Null:           -5008.0
Converged:              1.0000           Scale:             1.0000
---------------------------------------------------------------------
                       Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
---------------------------------------------------------------------
Loan Rate              6.1639   0.6946   8.8738 0.0000  4.8025  7.5254
Original Loan Size     0.0000   0.0000   7.6096 0.0000  0.0000  0.0001
Tier 2                -2.8078   0.2766 -10.1506 0.0000 -3.3499 -2.2656
Tier 3                -2.2206   0.2009 -11.0542 0.0000 -2.6143 -1.8269
Tier 4                -1.8535   0.1938  -9.5651 0.0000 -2.2333 -1.4737
Tier 5                -1.3482   0.1910  -7.0605 0.0000 -1.7225 -0.9740
Tier 6                -1.5777   0.2275  -6.9355 0.0000 -2.0235 -1.1318
Tier 7                -0.9574   0.2187  -4.3775 0.0000 -1.3861 -0.5288
Tier 8                -0.6750   0.2417  -2.7924 0.0052 -1.1487 -0.2012
Untiered              -0.7137   0.6361  -1.1220 0.2619 -1.9605  0.5330
Self-employed          0.2095   0.0842   2.4886 0.0128  0.0445  0.3745
Other                 -0.1588   0.1815  -0.8751 0.3815 -0.5145  0.1969
OriginalTermBucket    -0.0396   0.0028 -14.1184 0.0000 -0.0451 -0.0341
LTVBucket              0.9402   0.1715   5.4808 0.0000  0.6040  1.2764
MonthlyInstalmentpct  -0.8126   0.0576 -14.1021 0.0000 -0.9255 -0.6996
=====================================================================

Loan Rate               6.163935
Original Loan Size      0.000041
Tier 2                 -2.807756
Tier 3                 -2.220580
Tier 4                 -1.853543
Tier 5                 -1.348238
Tier 6                 -1.577679
Tier 7                 -0.957448
Tier 8                 -0.674971
Untiered               -0.713740
Self-employed           0.209474
Other                  -0.158814
OriginalTermBucket     -0.039582
LTVBucket               0.940177
MonthlyInstalmentpct   -0.812586
dtype: float64
```

In [497]:
```
# Select the parameters
X = df_final24.reset_index()[logisticParams]
```

```python
# Select the response valriable
y = df_final24.reset_index()['Default24']

# Apply the logistic regression model from statsmodels
model = sm.Logit(y, X)
logistic24 = model.fit()
print(logistic24.summary2())
print(logistic24.params)
```

```
Optimization terminated successfully.
         Current function value: 0.193461
         Iterations 9
                           Results: Logit
=====================================================================
Model:                Logit            No. Iterations:    9.0000
Dependent Variable:   Default24        Pseudo R-squared:  0.043
Date:                 2018-10-14 17:34 AIC:               13108.7122
No. Observations:     33802            BIC:               13235.1363
Df Model:             14               Log-Likelihood:    -6539.4
Df Residuals:         33787            LL-Null:           -6836.0
Converged:            1.0000           Scale:             1.0000
---------------------------------------------------------------------
                      Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
---------------------------------------------------------------------
Loan Rate             6.9268   0.5780  11.9834 0.0000  5.7939  8.0597
Original Loan Size    0.0000   0.0000   6.9858 0.0000  0.0000  0.0000
Tier 2               -2.9373   0.2496 -11.7666 0.0000 -3.4265 -2.4480
Tier 3               -2.0834   0.1730 -12.0453 0.0000 -2.4224 -1.7444
Tier 4               -1.7842   0.1686 -10.5801 0.0000 -2.1147 -1.4537
Tier 5               -1.3130   0.1674  -7.8439 0.0000 -1.6410 -0.9849
Tier 6               -1.5489   0.1954  -7.9264 0.0000 -1.9319 -1.1659
Tier 7               -0.9743   0.1899  -5.1295 0.0000 -1.3466 -0.6020
Tier 8               -0.7138   0.2086  -3.4217 0.0006 -1.1226 -0.3049
Untiered             -0.7810   0.5601  -1.3943 0.1632 -1.8788  0.3168
Self-employed         0.1415   0.0712   1.9873 0.0469  0.0019  0.2811
Other                -0.0173   0.1374  -0.1260 0.8997 -0.2865  0.2519
OriginalTermBucket   -0.0353   0.0023 -15.0995 0.0000 -0.0399 -0.0307
LTVBucket             0.6035   0.1392   4.3356 0.0000  0.3307  0.8763
MonthlyInstalmentpct -0.6580   0.0459 -14.3263 0.0000 -0.7480 -0.5680
=====================================================================

Loan Rate               6.926804
Original Loan Size      0.000034
Tier 2                 -2.937258
Tier 3                 -2.083372
Tier 4                 -1.784209
Tier 5                 -1.312956
Tier 6                 -1.548897
```

```
Tier 7                  -0.974333
Tier 8                  -0.713764
Untiered                -0.780970
Self-employed            0.141530
Other                   -0.017306
OriginalTermBucket      -0.035327
LTVBucket                0.603464
MonthlyInstalmentpct    -0.657991
dtype: float64
```

In the above summaries, we see that all parameters are statistically significant.

```
In [498]: # Select the parameters
          X = df_final12.reset_index()[logisticParams]

          # Select the response variable
          y = df_final12.reset_index()['Status 12']
```

Fit the Logistic Regression model

```
In [499]: lr12 = LogisticRegression(fit_intercept=True)
          lr12.fit(X,y)
```

```
Out[499]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                    penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                    verbose=0, warm_start=False)
```

The coefficients of the model are

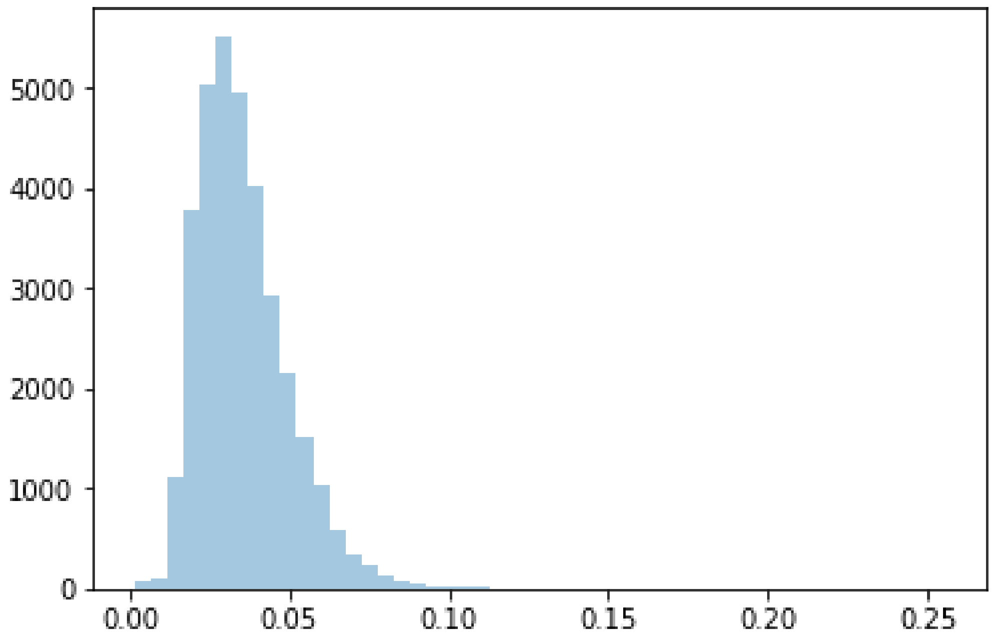
```
In [500]: print(lr12.coef_)
          print(lr12.intercept_)

          def Predict(df,classifier):
              return np.array(list(map(lambda x: 1 if x[0] > 0.97855 else False ,classifier.pre
```

```
[[ 1.59976095e-01  2.92372458e-05 -4.54305740e-01 -6.15464271e-01
  -3.81312261e-01  1.55597216e-01  5.40086770e-02  5.30806496e-01
   3.21431006e-01  1.08024664e-02  7.81397640e-02 -6.23405405e-02
  -3.31943323e-02  6.52889904e-02 -5.40578578e-01]]
[-0.46518357]
```

At this point we can use the predictions from the fitted model to get the probability distribution of the test data set. We see that the probability distribution of the test dataset is left skewed. This probability distribution is for the probability of default and as expected, defaulting is a rare event.

```
In [501]: # Probability distribution of default for the dataset
          sns.distplot(lr12.predict_proba(X)[:,1],kde=False)
          print('The mean of this distribution is {}'.format(np.mean(lr12.predict_proba(X)[:,1

          a = list(map(lambda x: round(x,3),lr12.predict_proba(X)[:,1]))
          print('The median of this distribution is {}'.format(statistics.median(a)))

          print('The IQR of this distribution is {}'.format(stats.iqr(a,rng=(25,75))))
```

```
The mean of this distribution is 0.035513834798503834
The median of this distribution is 0.033
The IQR of this distribution is 0.017999999999999995
```



We can then calculate the default probability of the consumers with the following profiles (person1 has in fact defaulted within the 12 month period but person2 hasn't). The main differences between these two consumers is the Tier and Loan Rate. Consumer1 is in Tier 5 whereas consumer2 is in Tier 3.

```
In [502]: person1 = df_final12[logisticParams].iloc[42,0:]

          person2 = df_final12[logisticParams].iloc[38,0:]

In [503]: print('Consumer1\n\n',person1,'\n\n','Consumer2\n\n',person2)
```

```
Consumer1
```

```
 Loan Rate                    0.173690
Original Loan Size      2441.177116
Tier 2                     0.000000
Tier 3                     0.000000
Tier 4                     0.000000
Tier 5                     1.000000
Tier 6                     0.000000
Tier 7                     0.000000
Tier 8                     0.000000
Untiered                   0.000000
Self-employed              0.000000
Other                      0.000000
OriginalTermBucket        35.000000
LTVBucket                  1.000000
MonthlyInstalmentpct       3.586794
Name: 43, dtype: float64

 Consumer2

 Loan Rate                    0.076448
Original Loan Size      2528.296369
Tier 2                     0.000000
Tier 3                     1.000000
Tier 4                     0.000000
Tier 5                     0.000000
Tier 6                     0.000000
Tier 7                     0.000000
Tier 8                     0.000000
Untiered                   0.000000
Self-employed              0.000000
Other                      0.000000
OriginalTermBucket        45.000000
LTVBucket                  0.900000
MonthlyInstalmentpct       3.234589
Name: 39, dtype: float64
```

The fitted logistic regression model has the following form:

$$logOdds = \ln(\frac{P(D)}{1-P(D)}) = \beta_0 + \beta_1 X_1 + ... + \beta_n X_n$$

In the above model, the coefficient of Loan Rate and Loan Rateˆ2 is 0.09 and 0.04 respectively. This means that increasing the loan rate is correlated with an increase in the probability of default.

By taking the exponential of both sides of this equation we can retrieve the probability of default (D):

$$P(D) = \frac{\exp(\beta_0 + \beta_1 X_1 + ... + \beta_n X_n)}{1 + \exp(\beta_0 + \beta_1 X_1 + ... + \beta_n X_n)}$$

Programmatically, this is as follows:

```
In [505]: logprob = lr12.intercept_[0]
          for i in range(0,len(logisticParams)):
              logprob += lr12.coef_[0][i]*person1[i]
          print('Person1 has a default probability of {}'.format((math.e**logprob)/(1+math.e**l

          logprob = lr12.intercept_[0]
          for i in range(0,len(logisticParams)):
              logprob += lr12.coef_[0][i]*person2[i]
          print('Person2 has a default probability of {}'.format((math.e**logprob)/(1+math.e**l

Person1 has a default probability of 0.0374750850615914
Person2 has a default probability of 0.015097223969645582
```

We can utilise the *predict_proba* method of the *Logistic Regression* object in order to get these values as well:

```
In [506]: person1 = person1.values.reshape(1,-1)
          person2 = person2.values.reshape(1,-1)

In [507]: print('Default probability of Consumer1 = {}'.format(lr12.predict_proba(person1)[0][1
          print('Default probability of Consumer2 = {}'.format(lr12.predict_proba(person2)[0][1

Default probability of Consumer1 = 0.0374750850615914
Default probability of Consumer2 = 0.01509722396964558
```

We do the same for the probability of default within the first 24 months

```
In [508]: y = df_final24.reset_index()['Default24']
          lr24 = LogisticRegression(fit_intercept=True)
          lr24.fit(X,y)
          print(lr24.coef_)
          print(lr24.intercept_)

          # Probability distribution of default for the dataset
          sns.distplot(lr24.predict_proba(X)[:,1],kde=False)
          print('The mean of this distribution is {}'.format(np.mean(lr24.predict_proba(X)[:,1]

          a = list(map(lambda x: round(x,3),lr24.predict_proba(X)[:,1]))
          print('The median of this distribution is {}'.format(statistics.median(a)))

          print('The IQR of this distribution is {}'.format(stats.iqr(a,rng=(25,75))))

          print('Default probability of Consumer1 = {}'.format(lr24.predict_proba(person1)[0][1
          print('Default probability of Consumer2 = {}'.format(lr24.predict_proba(person2)[0][1
```