Customer Churn Prediction

A Bank wants to take care of customer retention for their product; savings accounts. The bank wants you to identify customers likely to churn balances below the minimum balance. You have the customers information such as age, gender, demographics along with their transactions with the bank. Your task as a data scientist would be to predict the propensity to churn for each customer.

Each row represents a customer, each column contains attributes related to customer demographics and previous transactions with the bank.

Data Dictionary

There are multiple variables in the dataset which can be cleanly divided in 3 categories:

Demographic information about customers

customer_id - Customer id

vintage - Vintage of the customer with the bank in number of days

age - Age of customer

gender - Gender of customer

dependents - Number of dependents

occupation - Occupation of the customer

city- City of customer (anonymised)

Bank Related Information for customers

customer_nw_category - Net worth of customer (3:Low 2:Medium 1:High)

branch_code - Branch Code for customer account

days_since_last_transaction - No of Days Since Last Credit in Last 1 year

Transactional Information

current_balance - Balance as of today

previous_month_end_balance - End of Month Balance of previous month

average_monthly_balance_prevQ- Average monthly balances (AMB) in Previous Quarter

average_monthly_balance_prevQ2- Average monthly balances (AMB) in previous to previous quarter

percent_change_credits - Percent Change in Credits between last 2 quarters

current_month_credit - Total Credit Amount current month

previous_month_credit - Total Credit Amount previous month

current_month_debit - Total Debit Amount current month

previous_month_debit - Total Debit Amount previous month

current_month_balance - Average Balance of current month

previous_month_balance - Average Balance of previous month

churn - Average balance of customer falls below minimum balance in the next quarter (1/0).

Wow, we can already see there are many features in the data dictionary which we included in our hypothesis.

Loading Packages

Let us load the packages needed for visualization and exploratory analysis.

```
from google.colab import files
uploaded=files.upload()
```

```
Choose Files churn_prediction.csv
• churn_prediction.csv(application/vnd.ms-excel) - 3444690 bytes, last modified: 6/14/2019 - 100% done
Saving churn_prediction.csv to churn_prediction (1).csv
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
sns.set(style="white")
import io
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarnir
  import pandas.util.testing as tm
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold, StratifiedKFold, train_test_split
```

```
from sklearn.metrics import roc_auc_score, accuracy_score, confusion_matrix, roc_curve, pr

from sklearn.ensemble import RandomForestClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import classification_report
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
```

Loading Data

```
df= pd.read_csv(io.BytesIO(uploaded["churn_prediction.csv"]))
```

Exploratory Data Analysis

```
df.shape, df.columns
```

```
((28382, 21),
 Index(['customer_id', 'vintage', 'age', 'gender', 'dependents', 'occupation',
        'city', 'customer_nw_category', 'branch_code',
        'days_since_last_transaction', 'current_balance',
        'previous_month_end_balance', 'average_monthly_balance_prevQ',
        'average_monthly_balance_prevQ2', 'current_month_credit',
        'previous_month_credit', 'current_month_debit', 'previous_month_debit',
        'current_month_balance', 'previous_month_balance', 'churn'],
       dtype='object'))
```

We have data for 28382 customers with 21 columns. So, essentially we have 20 features and 1 target column which is churn. Let us quickly look at the values for each column.

```
df.iloc[1,:]
```

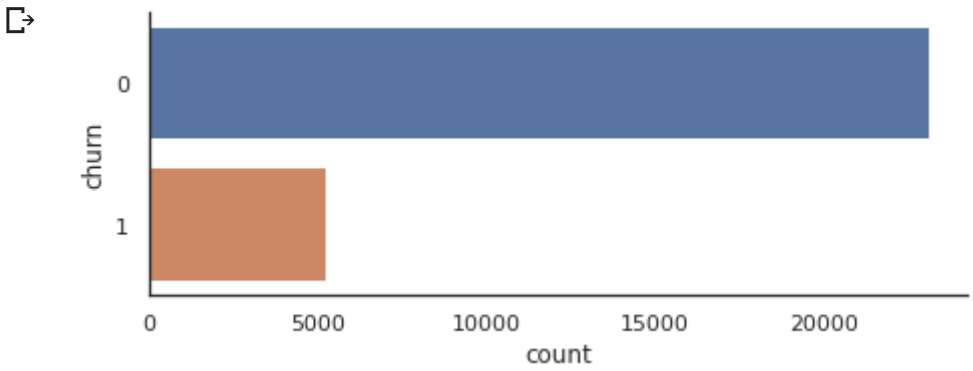
```
customer_id                    2
vintage                      310
age                           35
gender                      Male
dependents                     0
occupation          self_employed
city                          NaN
customer_nw_category           2
```

Alright. Here, we have a mix of categorical, numerical and ordinal variables as shown. There are missing values also in some of the features and we would treat them as a part of preprocessing step when we build the model.

Target Exploration

We are trying to predict if the client's account balance drops below the minimum balance prescribed for the customer. Clearly, this is a binary classification problem. Let's look at the target variable and find out how many customers are in the churn category.

```
ax = sns.catplot(y="churn", kind="count", data=df, height=2.6, aspect=2.5, orient='h')
```




```
df['churn'].value_counts(normalize = True)
```

```
0    0.814671
1    0.185329
Name: churn, dtype: float64
```

Numerical features

Let us look at the numerical features. From the description provided in the data dictionary and cell 8, we can see that we have the following numerical features. Let us quickly describe them to check the following:

Count: Can be used to check for missing value count

Mean: Mean of the variable

Standard Deviation: Standard deviation of the variable

Minimum: Minimum value of the variable

Quantile values: 25, 50 (median) & 75% quantiles of the variable

Maximum: Maximum value of the variable.

Notice that we will not directly used dtypes function to identify numerical columns but rather used business sense to select numerical features as we have seen from a smaple record, branch code and city code actually represent categories and not some meaningful numerical value.

```
df.dtypes
```

```
customer_id                        int64
vintage                            int64
age                                int64
gender                            object
dependents                       float64
occupation                        object
city                             float64
customer_nw_category               int64
branch_code                        int64
days_since_last_transaction      float64
current_balance                  float64
previous_month_end_balance       float64
average_monthly_balance_prevQ    float64
average_monthly_balance_prevQ2   float64
current_month_credit             float64
previous_month_credit            float64
current_month_debit              float64
previous_month_debit             float64
current_month_balance            float64
previous_month_balance           float64
churn                              int64
dtype: object
```

As shown dtypes function puts city and branch code features in the numerical category but that is not the intention.

```
numerical_cols = ['customer_id', 'vintage', 'age', 'dependents', 'customer_nw_category', '
       'previous_month_end_balance', 'average_monthly_balance_prevQ', 'average_monthly_bal
       'current_month_credit','previous_month_credit', 'current_month_debit', 'previous_mo
       'current_month_balance', 'previous_month_balance']
df[numerical_cols].describe()
```

contributed by the churning customers.

- It might be a good idea to create a feature which is the difference of these 2 variables during the model building process.
- Students are encouraged to do more univariate analysis on other balances and check distributions to find similar insights.

Next, in order to understand which of these features might be important to predict the churn, we will do a bivariate analysis.

Bivariate Analysis

Now, we will check the relationship of the numeric variables along with the target. Again conversion to log is important here as we have a lot of outliers and visualization will be difficult for it.

Churn vs Current & Previous month balances

```
balance_cols = ['current_balance','previous_month_end_balance',
                'current_month_balance', 'previous_month_balance']
df1 = pd.DataFrame()

for i in balance_cols:
    df1[str('log_')+ i] = np.log(df[i] + 6000)

log_balance_cols = df1.columns


df1['churn'] = df['churn']
```

We will use the brilliant pairplot function from Seaborn which supports displaying relationship between multiple variables. It displays the scatter plot between a pair of feature and also displays the distribution.

Here I have included the following:

- Log of current balance & previous month end balance
- Log of average monthly balance of current and previous month
- Churn is represented by the color here **(Orange - Churn, Blue - Not Churn)**

```
#sns.pairplot(df1,vars=log_balance_cols, hue = 'churn',plot_kws={'alpha':0.1})
df1_no_churn = df1[df1['churn'] == 0]
sns.pairplot(df1_no_churn,vars=log_balance_cols,plot_kws={'alpha':0.1})
plt.show()
```

	customer_id	vintage	age	dependents	customer_nw_category	c
count	28382.000000	28382.000000	28382.000000	25919.000000	28382.000000	
mean	15143.508667	2364.336446	48.208336	0.347236	2.225530	
std	8746.454456	1610.124506	17.807163	0.997661	0.660443	

Lets list down a few key observations:

- Customer ID here is just an id variable identifying a unique customer and has values between 1 and 30301
- On an average, a customer from this set has been with the bank for 2400 days or around 6.5 years
- On an average, a customer has less than 1 dependent and has an average age of 48 years
- A general trend on variables which are related to balances have a wide range with huge outliers, it will key to observe these outliers
- Most of the customers lie in category 2 or 3 for net worth and have on an average done the last transaction 70 days ago. Now the high net worth customers (Category) must have high credit, debit and balance values. Let's verify this using data.

Customer Net worth Category & Balance Features

We will use a groupby function to check the mean values of balance features.

```
cols = ['current_balance',
       'previous_month_end_balance', 'average_monthly_balance_prevQ', 'average_monthly_bal
        'current_month_credit','previous_month_credit', 'current_month_debit', 'previous_m
       'current_month_balance', 'previous_month_balance']
df.groupby(['customer_nw_category'])[cols].mean()
```

customer_nw_category	current_balance	previous_month_end_balance	average_monthly_b
1	12883.682913	13436.613544	
2	7773.279345	7744.492544	
3	4795.520175	4957.438675	

So there is clear consistency here as mean values of balance features and the credit/debit features have higher values for net worth category 1 and lower value for net worth categories 2 & 3.

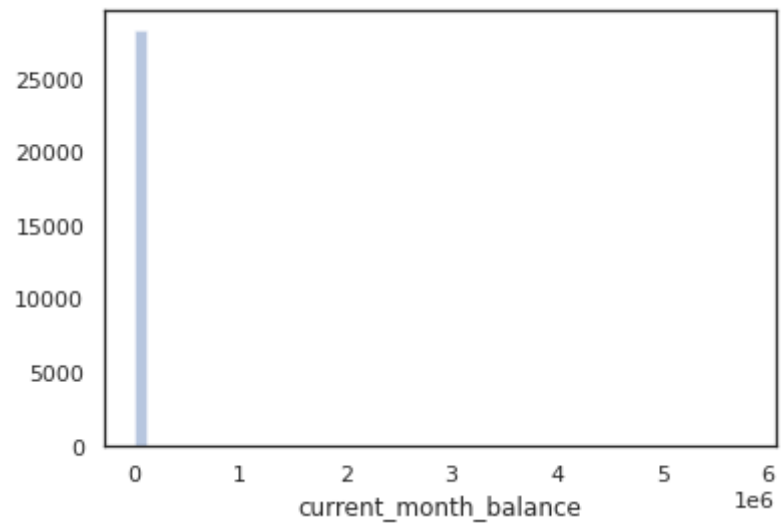
The bulk of features are comprised of balance and credit debit features. Let us explore them in detail.

Balance & Credit/Debit Features

We will start by looking at average balance in the current month. We will use a histogram to check its distribution.

Average Monthly Balance Features

```
sns.distplot(df['current_month_balance'], kde = False)
plt.show()
```



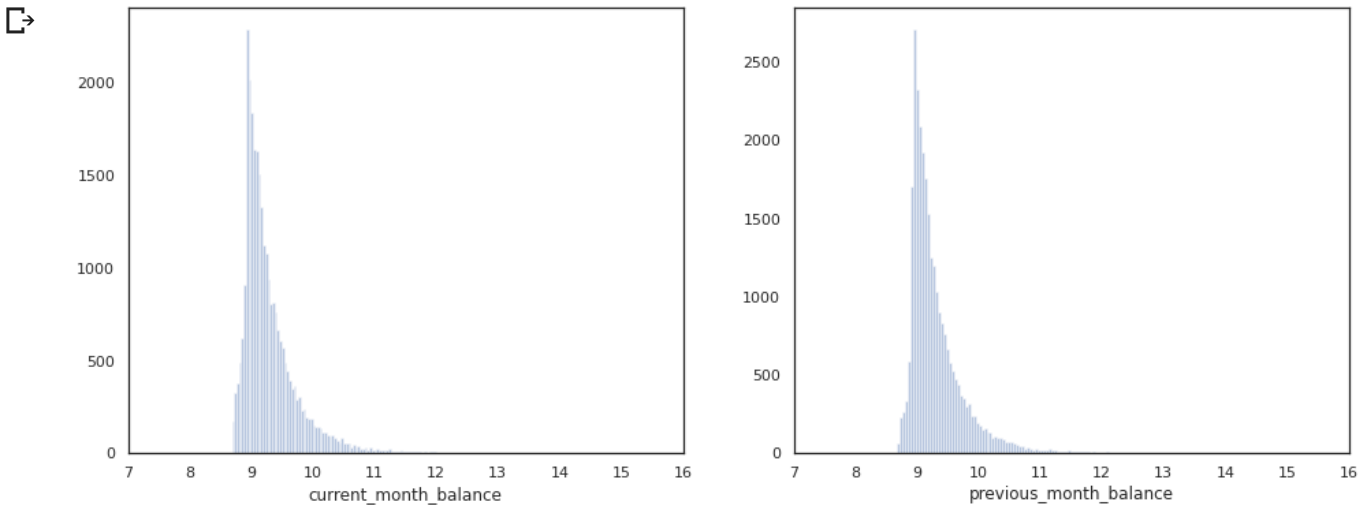
Due to the huge outliers in both positive and negative directions, it is very difficult to derive insights from this plot.

- In this case, we could convert such columns to log and then check the distributions.
- However, since there are negative values, it cannot be a direct log conversion as log of negative numbers is not defined.
- To tackle this, we add a positive constant within the log as a correction and to account for negative values we add a constant value within log

```
temp = np.log(df['current_month_balance'] + 6000)

sns.distplot(temp, kde = False, bins = 100)
plt.show()
```

Now, we can see more clearly that this is a right skewed feature and we have much more clarity on its distribution. Let us use subplot to quickly look at more numerical features together and see trends.

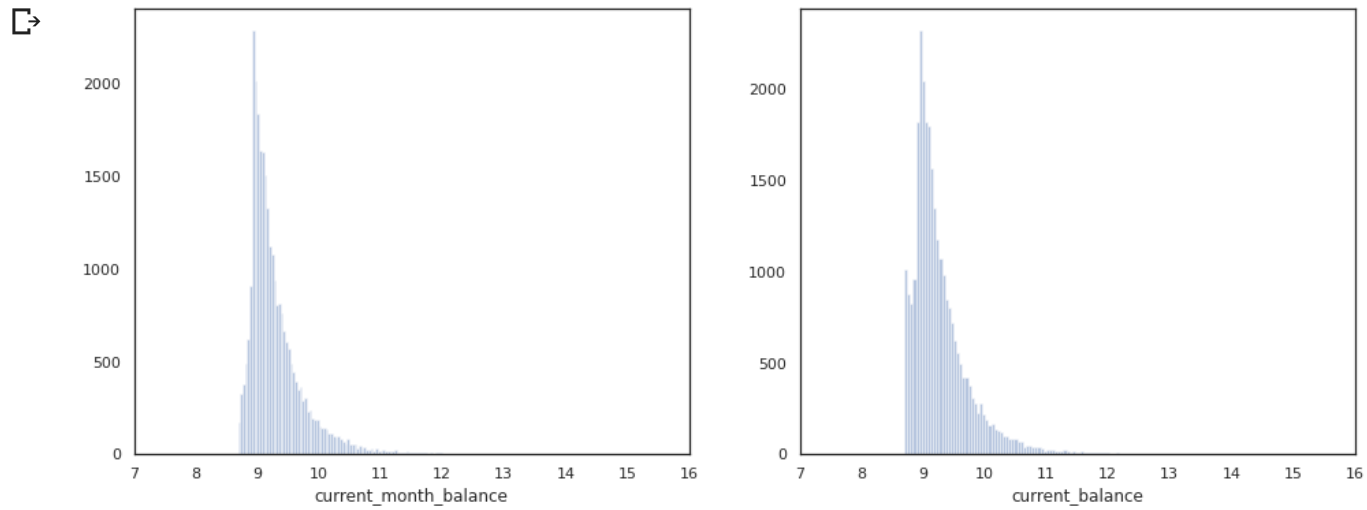
```
# Numerical Features
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(16, 6))
xmin = 7
xmax = 16
# Current Month Average Balance
temp = np.log(df['current_month_balance'] + 6000) # To account for negative values we add
ax1.set_xlim([xmin,xmax])
ax1.set(xlabel='log of average balance of current month')
sns.distplot(temp, kde = False, bins = 200, ax = ax1)


# Previous month average balance
temp = np.log(df['previous_month_balance'] + 6000) # To account for negative values we add
ax2.set_xlim([xmin,xmax])
ax2.set(xlabel='log of average balance of previous month')
sns.distplot(temp, kde = False, bins = 200, ax = ax2)


plt.show()
```



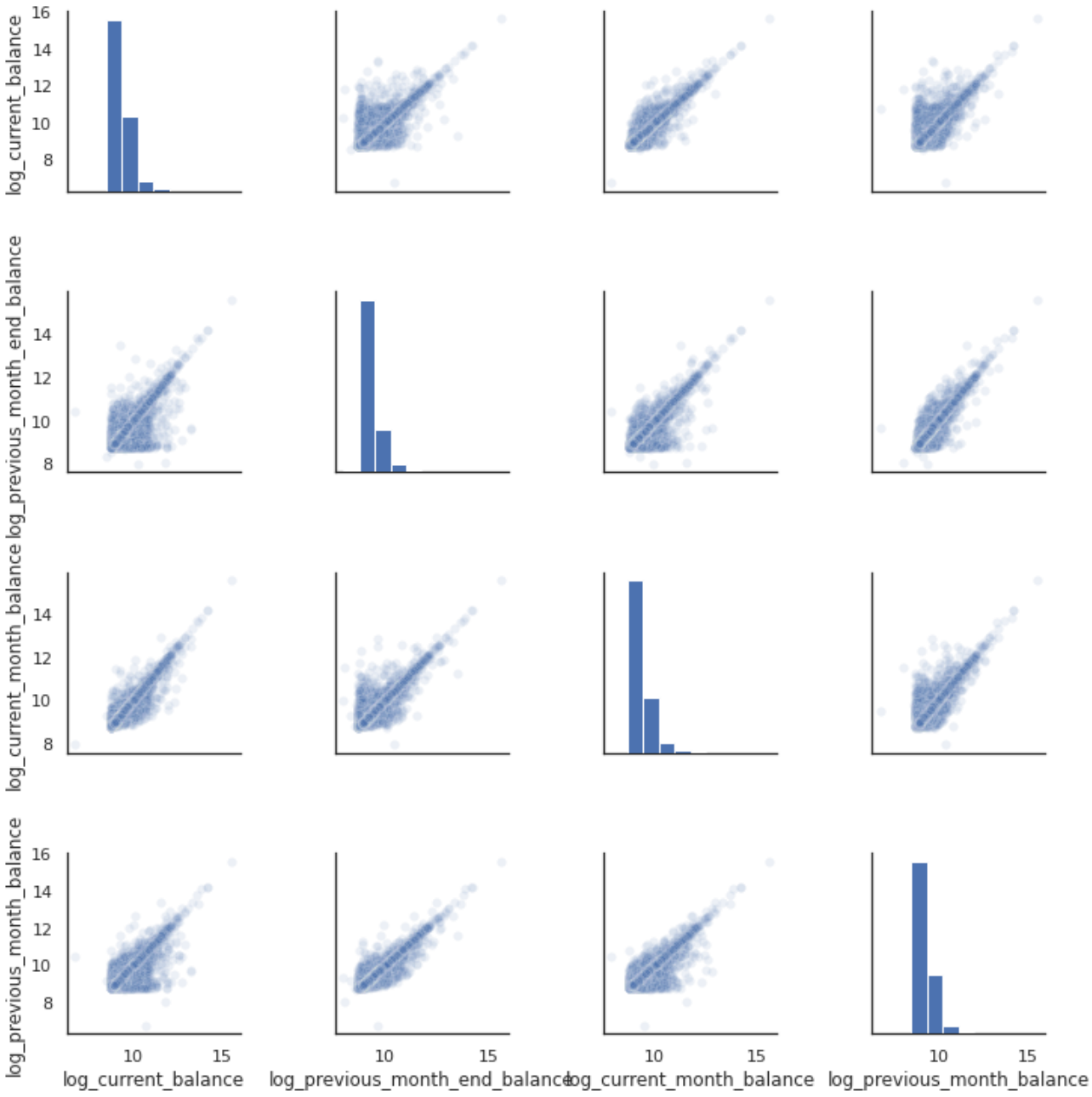
Current Balance today vs Average Monthly Balance in current month

As expected the average monthly balance for both months are quite similar and have right skewed histograms as shown. Now let us compare the current month average balance vs current balance as of today.

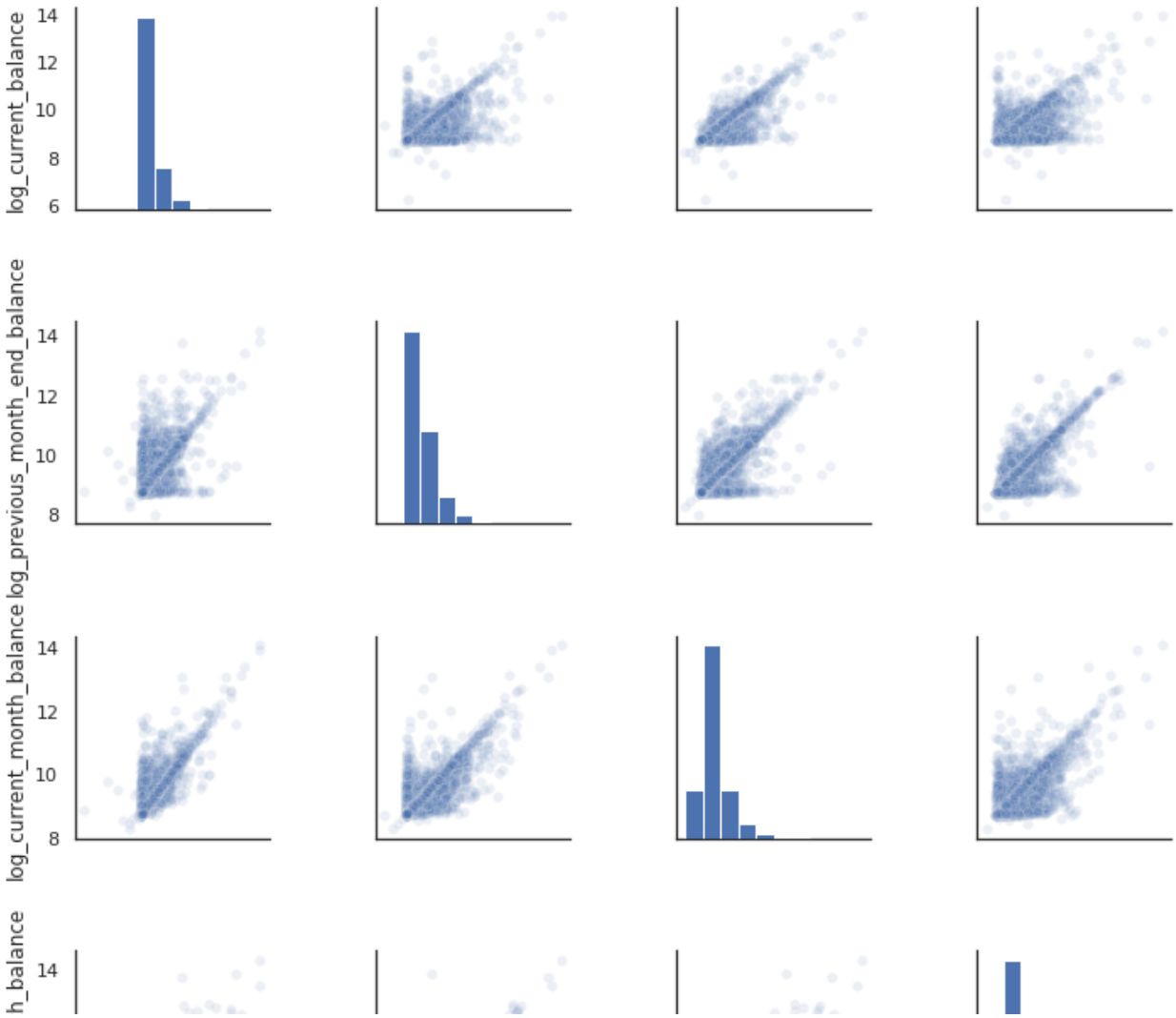
```
# Numerical Features
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(16, 6))
xmin = 7
xmax = 16
# Current Month Average Balance
temp = np.log(df['current_month_balance'] + 6000) # To account for negative values we add
ax1.set_xlim([xmin,xmax])
#ax1.set(xlabel='log of average balance of current month')
sns.distplot(temp, kde = False, bins = 200, ax = ax1)


# Current End of month average balance
temp = np.log(df['current_balance'] + 6000) # To account for negative values we add a cons
ax2.set_xlim([xmin,xmax])
#ax2.set(xlabel='log of month end balance of current  month')
sns.distplot(temp, kde = False, bins = 200, ax = ax2)


plt.show()
```



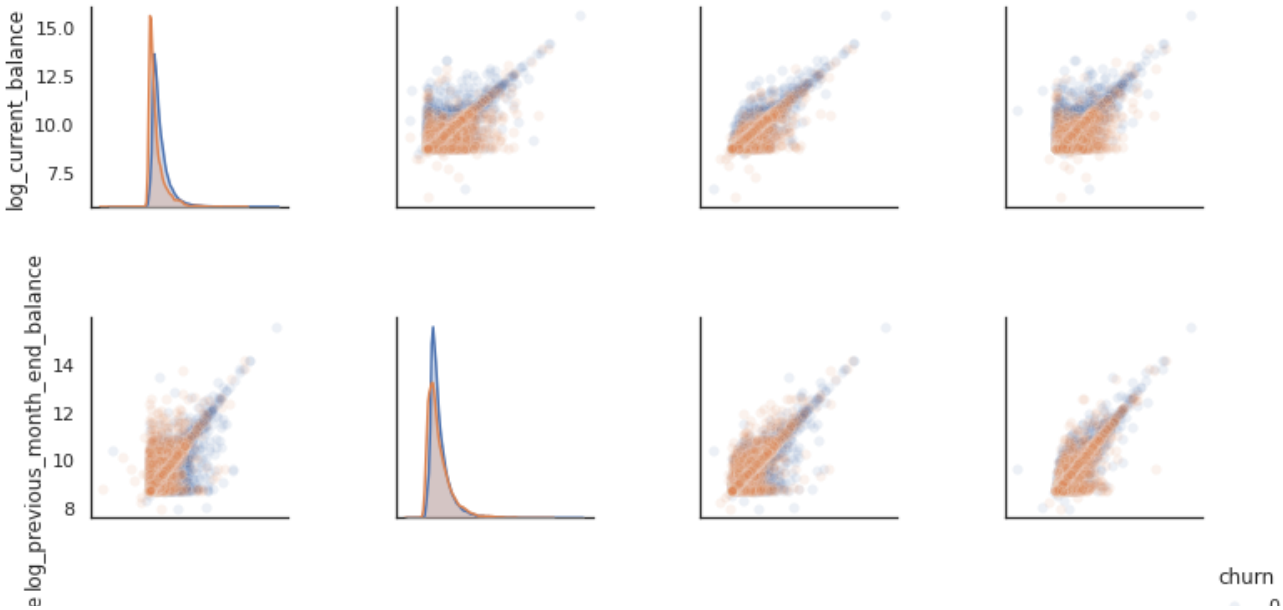
- Here, we can see that the distribution for both lie in almost the same interval, however, there are larger number of values for current balance just below 9 which might have been




```
#sns.pairplot(df1,vars=log_balance_cols, hue = 'churn',plot_kws={'alpha':0.1})
df1_churn = df1[df1['churn'] == 1]
sns.pairplot(df1_churn,vars=log_balance_cols,plot_kws={'alpha':0.1})
plt.show()
```




```
sns.pairplot(df1,vars=log_balance_cols,hue ='churn',plot_kws={'alpha':0.1})
plt.show()
```



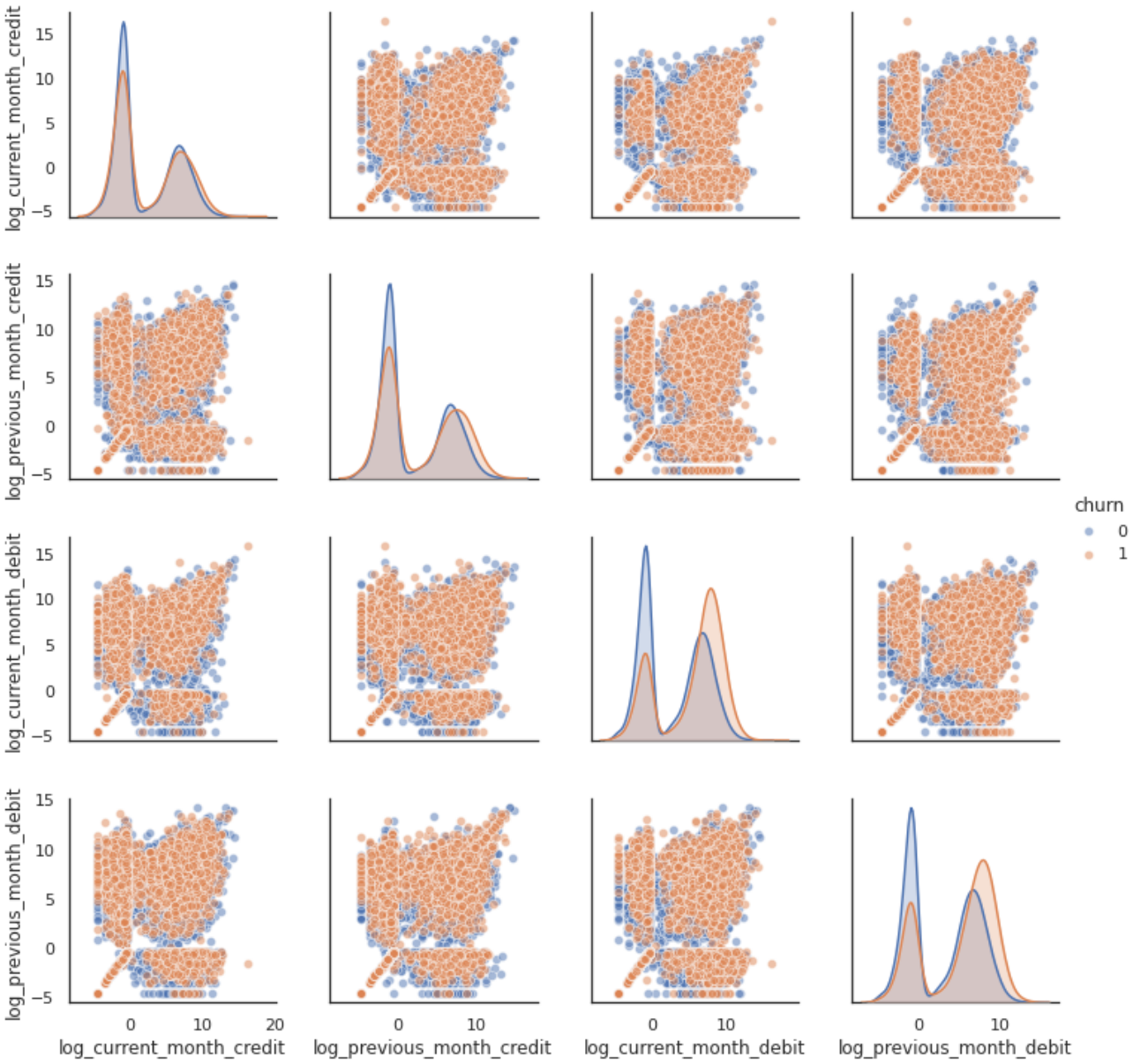
The distribution for these features look similar. We can make the following conclusions from this:

- There is high correlation between the previous and current month balances which is expected
- The lower balances tend to have higher number of churns which is clear from the scatter plots
- Distribution for the balances are all right skewed

Credit and Debits for current and previous months

Total credit and debit amounts for the current and previous can be clubbed into the same category. Let us again use the pair plot to check distributions and scatter plots.

```
cr_dr_cols = ['current_month_credit','previous_month_credit',
              'current_month_debit', 'previous_month_debit']
df1 = pd.DataFrame()

for i in cr_dr_cols:
    df1[str('log_')+ i] = np.log(df[i])

log_dr_cr_cols = df1.columns


df1['churn'] = df['churn']


sns.pairplot(df1,vars=log_dr_cr_cols, hue = 'churn',plot_kws={'alpha':0.5})
plt.show()
```



Both credit and debit patterns show significant difference in distributions for churned and non churned customers.

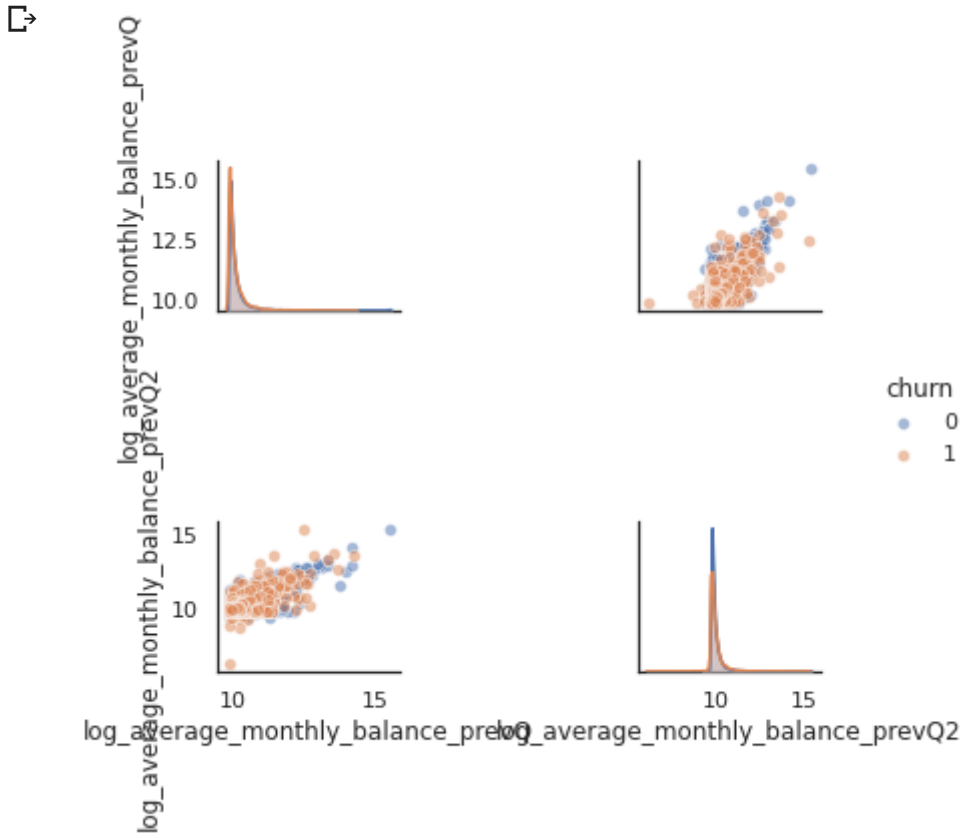
1. Bimodal distribution/Double Bell Curve shows that there are 2 different types of customers with 2 brackets of credit and debit. Now, during the modeling phase, these could be considered as a seperate set of customers
2. For debit values, we see that there is a significant difference in the distribution for churn and non churn and it might be turn out to be an important feature

Average monthly balance of previous and previous to previous quarters

Now, these 2 variables present deeper historical transactions and would help in understanding the trend during the last 2 quarters.

```
q_cols = ['average_monthly_balance_prevQ', 'average_monthly_balance_prevQ2']
df1 = pd.DataFrame()
```

```
for i in q_cols:
    df1[str('log_')+ i] = np.log(df[i] + 17000)

log_q_cols = df1.columns
df1['churn'] = df['churn']


sns.pairplot(df1,vars=log_q_cols, hue = 'churn',plot_kws={'alpha':0.5})
plt.show()
```



The distributions do not have much difference when it comes to churn.

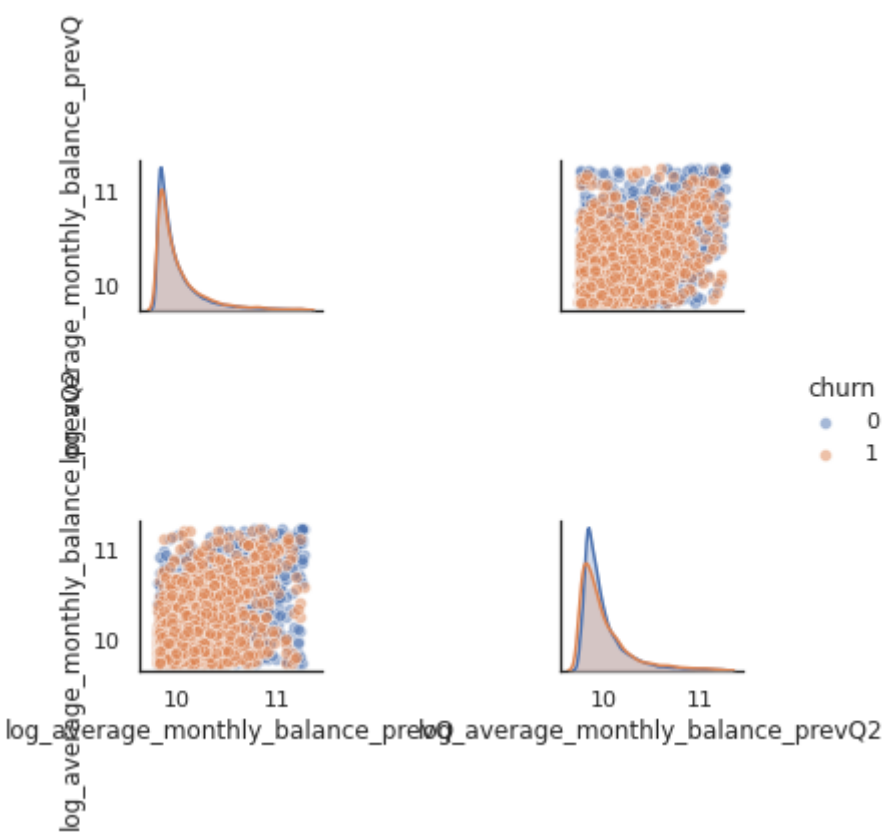
However, there are some high negative values in the previous to previous quarters due to which there appears to be a lateral shift. However, if you look at the x-axis, it is still at the same scale for both features.

Removing the extreme outliers from the data using the 1 and 99th percentile would help us look at the correct distributions

```
# Remove 1st and 99th percentile and plot

df2 = df[['average_monthly_balance_prevQ', 'average_monthly_balance_prevQ2']]

low = .01
high = .99
quant_df = df2.quantile([low, high])
print(quant_df)
```

```
       average_monthly_balance_prevQ  average_monthly_balance_prevQ2
0.01                        1449.0377                        121.6485
0.99                       60118.2288                      59357.8810
```

```
df3 = df2.apply(lambda x: x[(x>quant_df.loc[low,x.name]) &
                                    (x < quant_df.loc[high,x.name])], axis=0)


q_cols = ['average_monthly_balance_prevQ', 'average_monthly_balance_prevQ2']
df1 = pd.DataFrame()

for i in q_cols:
    df1[str('log_')+ i] = np.log(df3[i] + 17000)

log_q_cols = df1.columns
df1['churn'] = df['churn']


sns.pairplot(df1,vars=log_q_cols, hue = 'churn',plot_kws={'alpha':0.5})
plt.show()
```



Now, we can clearly see that the distributions are very similar for both the variables and and non churning customers have higher average monthly balances in previous 2 quarters.

Demographics and Bank Related Information for customers

Revisiting the description for numerical demographic & Bank related customer variables we have:

Numerical:

vintage - Vintage of the customer with the bank in number of days

age - Age of customer

days_since_last_transaction - No of Days Since Last Credit in Last 1 year

Ordinal:

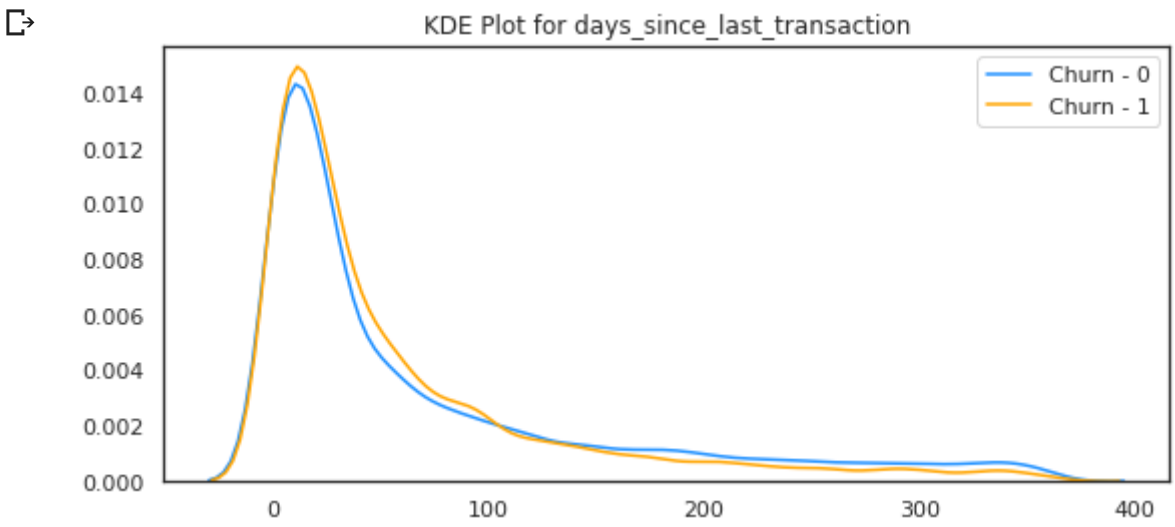
dependents - Number of dependents

customer_nw_category - Net worth of customer (3:Low 2:Medium 1:High)

KDE plot can be used for numerical variables on the same axis to quickly compare the distributions for churning and non churning customers. It basically plots the approximate churn rate against each normal variable. This is exactly similar to what we did in the pairplot with distributions but here we would look at them separately since they represent entirely different variables.

Days Since Last Transaction

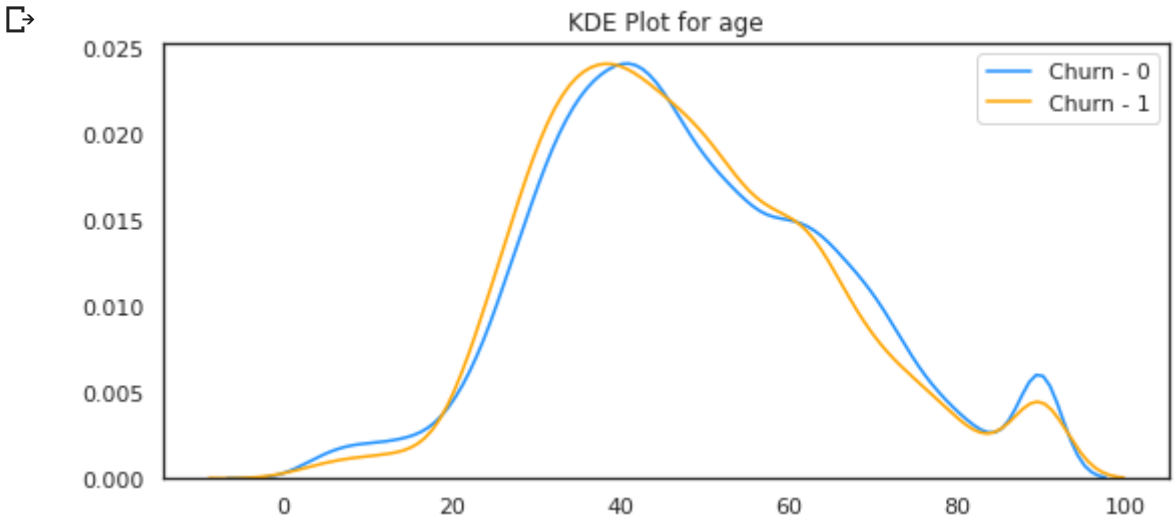
```
# KDE Plot Smoothens out even if there are no values for a value
def kdeplot(feature):
    plt.figure(figsize=(9, 4))
    plt.title("KDE Plot for {}".format(feature))
    ax0 = sns.kdeplot(df[df['churn'] == 0][feature].dropna(), color= 'dodgerblue', label=
    ax1 = sns.kdeplot(df[df['churn'] == 1][feature].dropna(), color= 'orange', label= 'Chu

kdeplot('days_since_last_transaction')
```



There is no significant difference between the distributions for churning and non churning customers when it comes to days since last transaction.

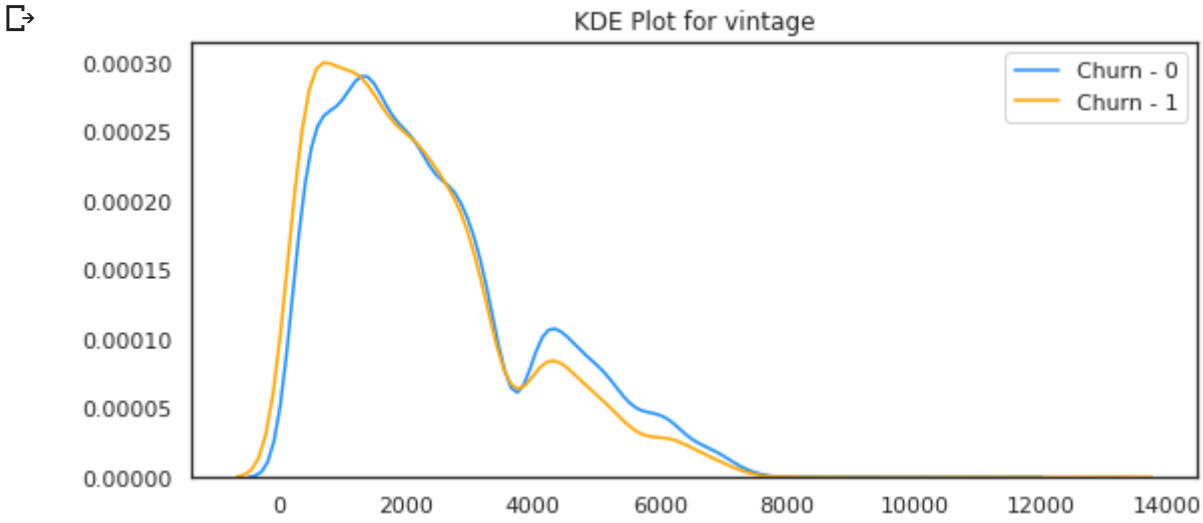
Age & Vintage

```
kdeplot('age')
```



Similarly, age also does not significantly affect the churning rate. However, customers above 80 years of age less likely to churn.

```
kdeplot('vintage')
```



For most frequent vintage values, the churning customers are slightly higher, while for higher values of vintage, we have mostly non churning customers which is in sync with the age variable.

Categorical features

This dataset has 4 categorical features (gender, occupation, city and branch code) as can be inferred from the data dictionary. Now let us have a look at the the number of unique values for each of them.

```
cat_cols = ['gender', 'occupation', 'city','branch_code']

for i in range(0,len(cat_cols)):
    print(str(cat_cols[i]) + " - Number of Unique Values: " + str(df[cat_cols[i]].nunique(
```

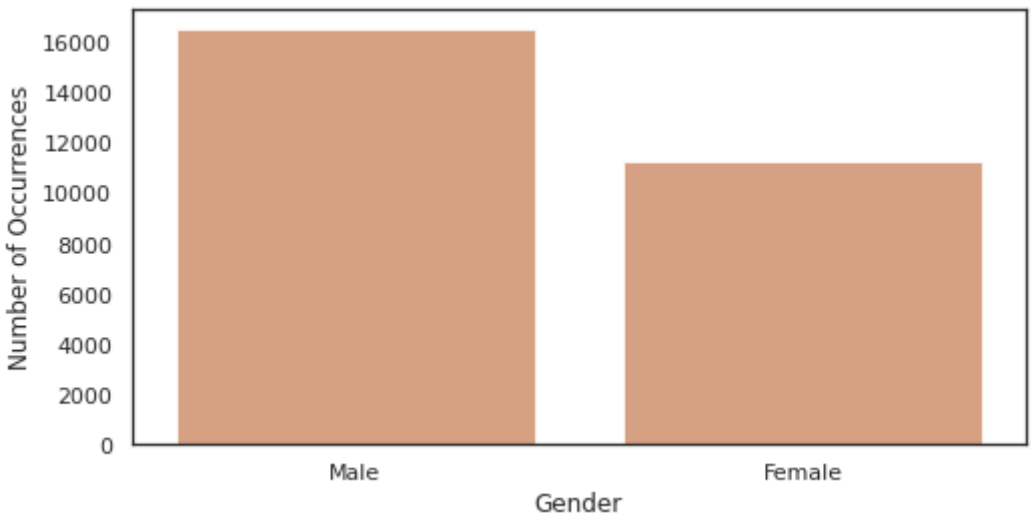
```
gender - Number of Unique Values: 2
occupation - Number of Unique Values: 5
city - Number of Unique Values: 1604
branch_code - Number of Unique Values: 3185
```

So, there are a large number of unique values for branch code and city. Gender has 2 unique values while occupation has 7.

Univariate Analysis

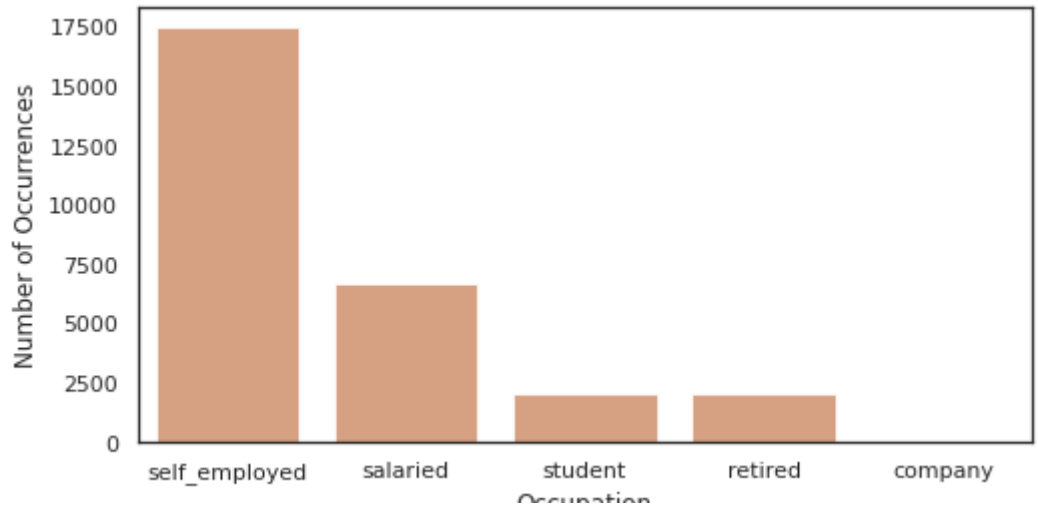
Let us look at each categorical feature and check distribution.

```
color = sns.color_palette()

int_level = df['gender'].value_counts()

plt.figure(figsize=(8,4))
sns.barplot(int_level.index, int_level.values, alpha=0.8, color=color[1])
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Gender', fontsize=12)
plt.show()
```



Amongst the customers, we have more males than females here. Lets check occupation now.

```
color = sns.color_palette()

int_level = df['occupation'].value_counts()

plt.figure(figsize=(8,4))
sns.barplot(int_level.index, int_level.values, alpha=0.8, color=color[1])
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Occupation', fontsize=12)
plt.show()
```

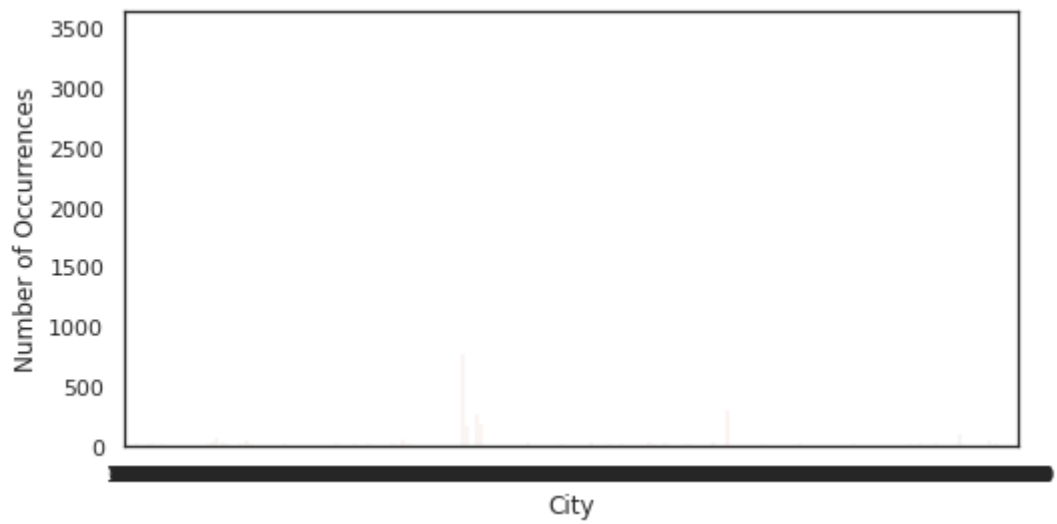


Most of the customers are self employed, followed by salaried account holders, retired custumors and very low number of companies.

Now, branch code and city code have a lot of unique values and direct visualization will be difficult. Lets see how:

City Code & Branch Code

```
color = sns.color_palette()

int_level = df['city'].value_counts()

plt.figure(figsize=(8,4))
sns.barplot(int_level.index, int_level.values, alpha=0.8, color=color[1])
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('City', fontsize=12)
plt.show()
```

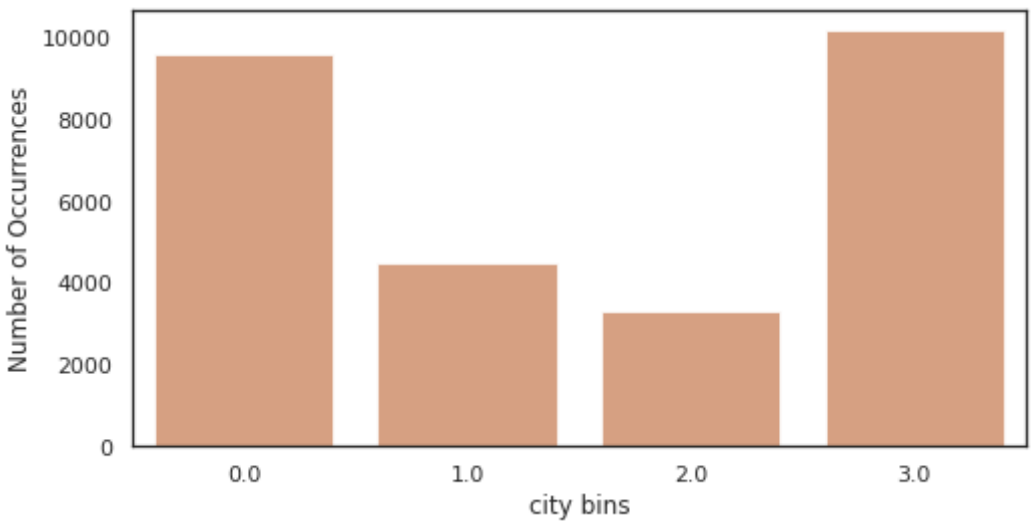


Now, let us have a look at the frequencies of the top city codes:

```
df['city'].value_counts().head(20)
```

```
1020.0    3479
1096.0    2016
409.0     1334
146.0     1291
834.0     1138
334.0      930
1232.0     840
623.0      778
15.0       669
575.0      631
1525.0     375
905.0      345
491.0      312
1111.0     312
1494.0     289
649.0      281
1589.0     251
318.0      245
1084.0     240
61.0       237
Name: city, dtype: int64
```

```
# Convert city variable wrt degree of number of customers
df['city_bin'] = df['city'].copy()
counts = df.city.value_counts()
df.city_bin[df['city'].isin(counts[counts > 900].index)] = 3
df.city_bin[df['city'].isin(counts[counts < 900].index) & df['city_bin'].isin(counts[count
df.city_bin[df['city'].isin(counts[counts < 350].index) & df['city_bin'].isin(counts[count
df.city_bin[df['city'].isin(counts[counts < 100].index)] = 0

df['city_bin'] = pd.to_numeric(df['city_bin'], errors='coerce')


int_level = df['city_bin'].value_counts()

plt.figure(figsize=(8,4))
sns.barplot(int_level.index, int_level.values, alpha=0.8, color=color[1])
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('city bins', fontsize=12)
plt.show()
```



There are 2 major categories here. Cities with more than 900 occurances and with less than 100 occurances. Similarly we can create bins for branch id and have a look.

```
df['branch_code'].value_counts()
```

```
19      145
6       142
60      128
16      111
8       109
       ...
3386      1
3418      1
4082      1
4050      1
2049      1
Name: branch_code, Length: 3185, dtype: int64
```

```
# Convert city variable wrt degree of number of customers
df['branch_bin'] = df['branch_code'].copy()
counts = df.branch_code.value_counts()
df.branch_bin[df['branch_code'].isin(counts[counts >= 100].index)] = 2
df.branch_bin[df['branch_code'].isin(counts[counts < 100].index) & df['branch_bin'].isin(c
df.branch_bin[df['branch_code'].isin(counts[counts < 50].index)] = 0

df['branch_bin'] = pd.to_numeric(df['branch_bin'], errors='coerce')
```

```
df['branch_bin'].value_counts()
```

```
0    23237
1     4405
2      740
Name: branch_bin, dtype: int64
```

```
int_level = df['branch_bin'].value_counts()

plt.figure(figsize=(8,4))
sns.barplot(int_level.index, int_level.values, alpha=0.8, color=color[1])
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('branch bins', fontsize=12)
plt.show()
```

So creating brackets/bins on the basis of frequency is a good idea to quickly analyse a variable with high number of categories.

Next, Let us look at some bivariate analysis for categorical variables.

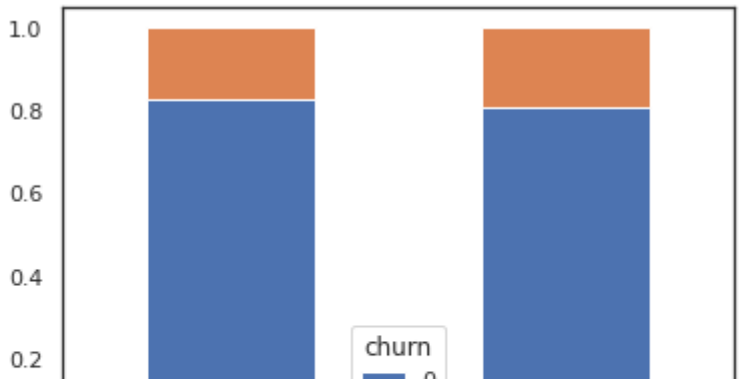
Bivariate Analysis

Lets define a function to quickly compare churn rates for different categories in each feature.

```
def barplot_percentages(feature):
    #fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(16, 6))
    ax1 = df.groupby(feature)['churn'].value_counts(normalize=True).unstack()
    ax1.plot(kind='bar', stacked='True')
    int_level = df[feature].value_counts()

    plt.figure(figsize=(8,4))
    sns.barplot(int_level.index, int_level.values, alpha=0.8, color=color[1])
    plt.ylabel('Number of Occurrences', fontsize=12)
    plt.xlabel(str(feature), fontsize=12)
    plt.show()
```

Gender

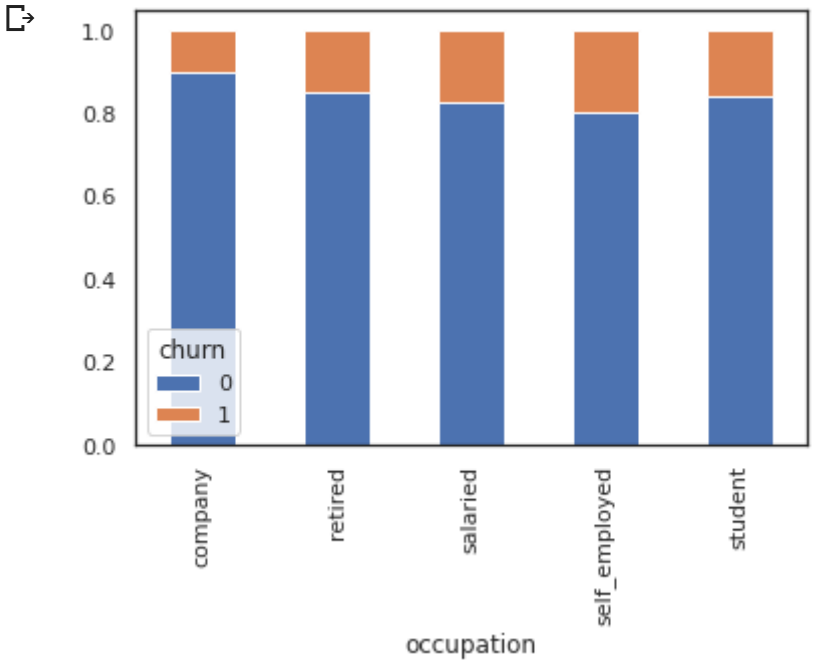
```
barplot_percentages("gender")
```

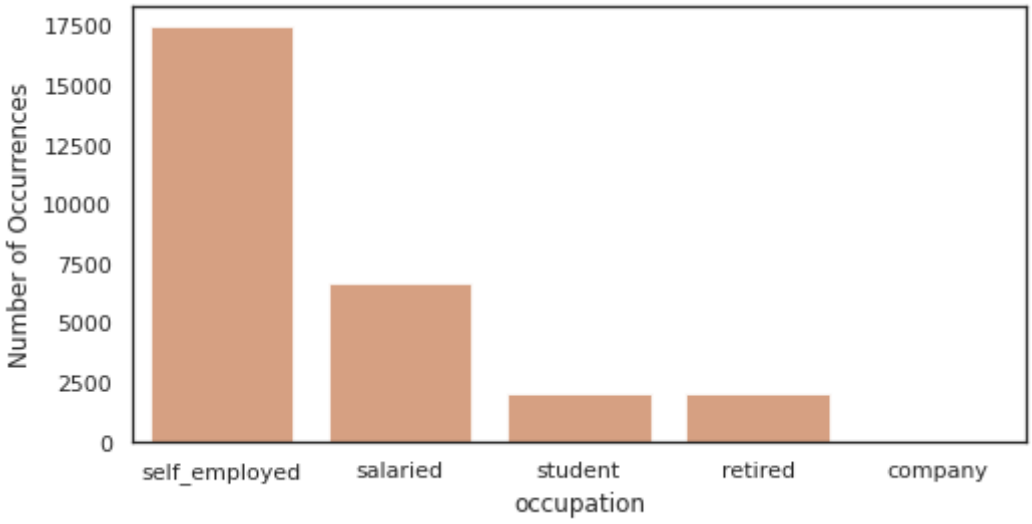


Does not look like a very significant variable as the ratio of churned customers and others is very similar.

Occupation


```
barplot_percentages("occupation")
```

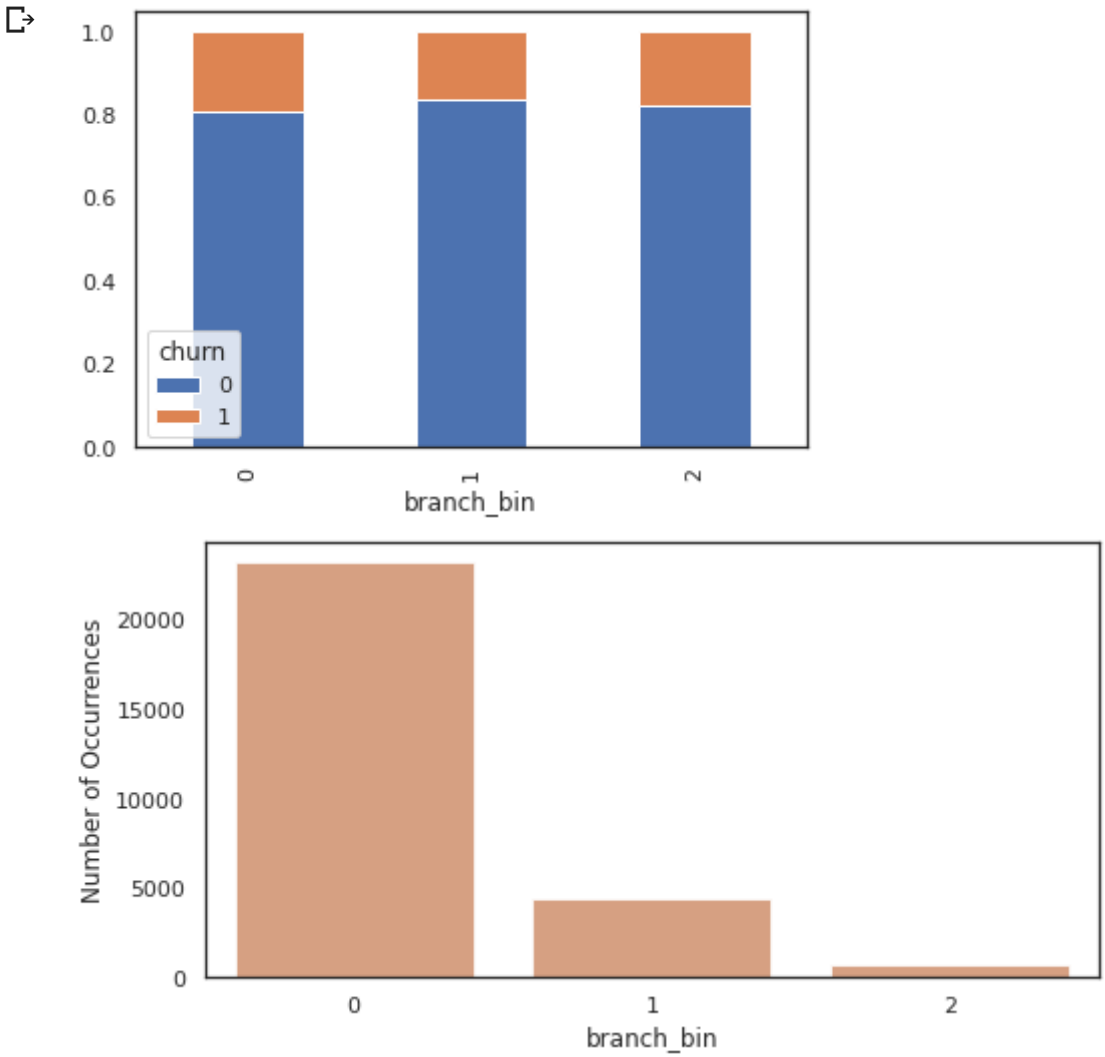




Self Employed and salaried have higher churn rate and are the major categories.

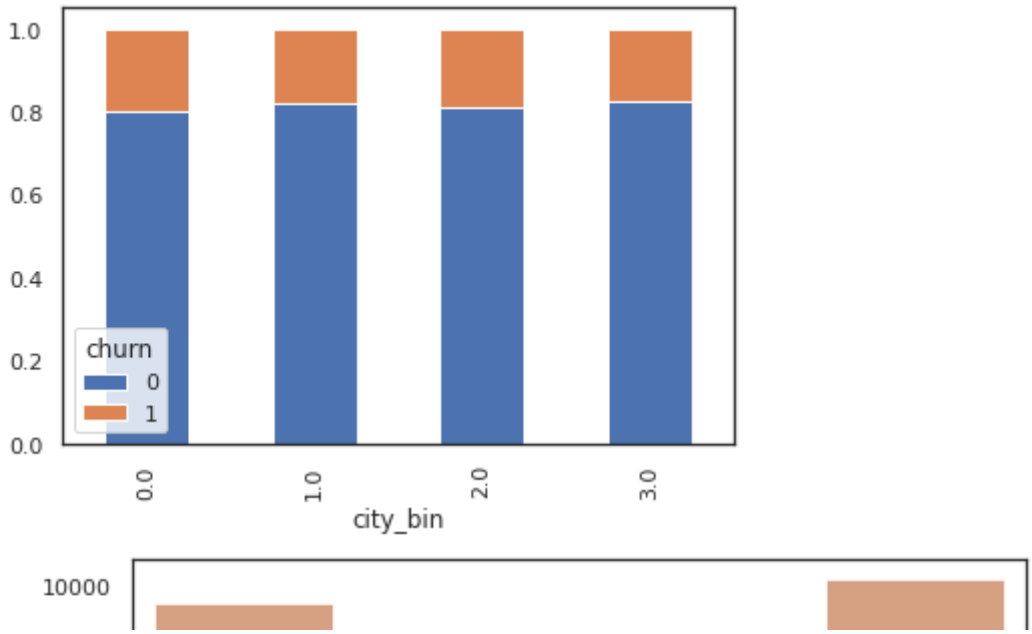
Branch Bins

```
barplot_percentages("branch_bin")
```



City Bins

```
barplot_percentages("city_bin")
```

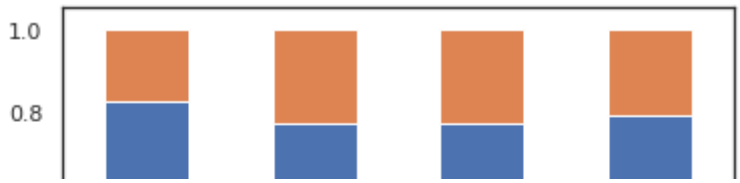


Here, we see significant difference for different occupations and certainly would be interesting to use as a feature for prediction of churn. However, city and branch codes have little difference amongst the different types of branches.

Dependents

```
df['dependents'][df['dependents'] > 3] = 3
```

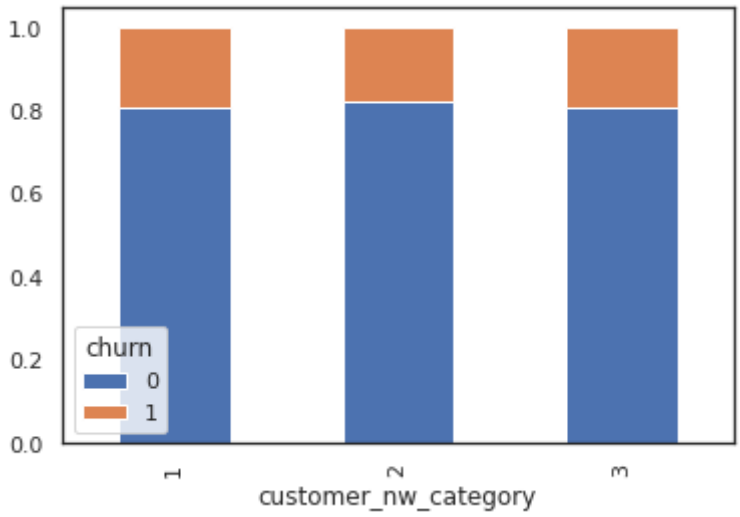
```
barplot_percentages("dependents")
```

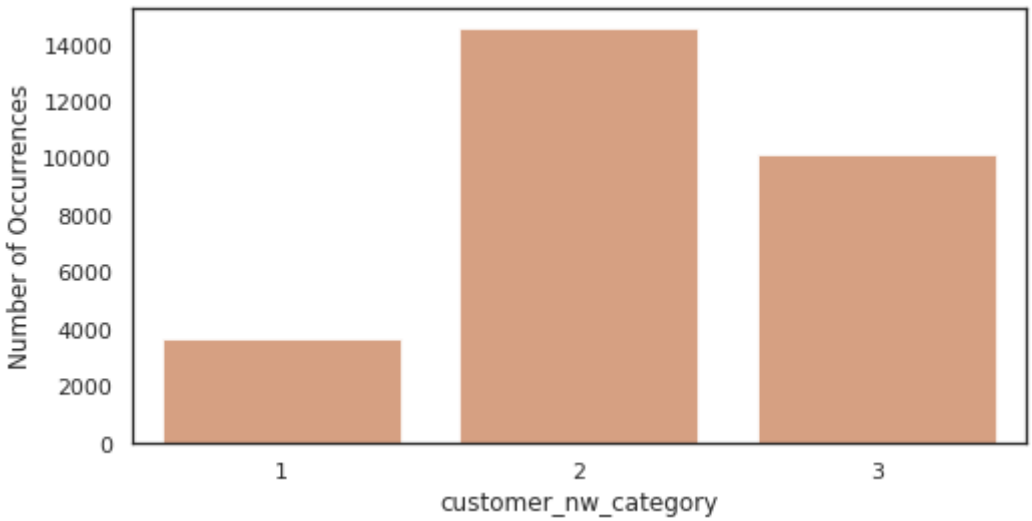


Most customers have no dependents and hence this variable in itself has low variance so it is of little significance.

Customer Net worth Category

```
barplot_percentages("customer_nw_category")
```





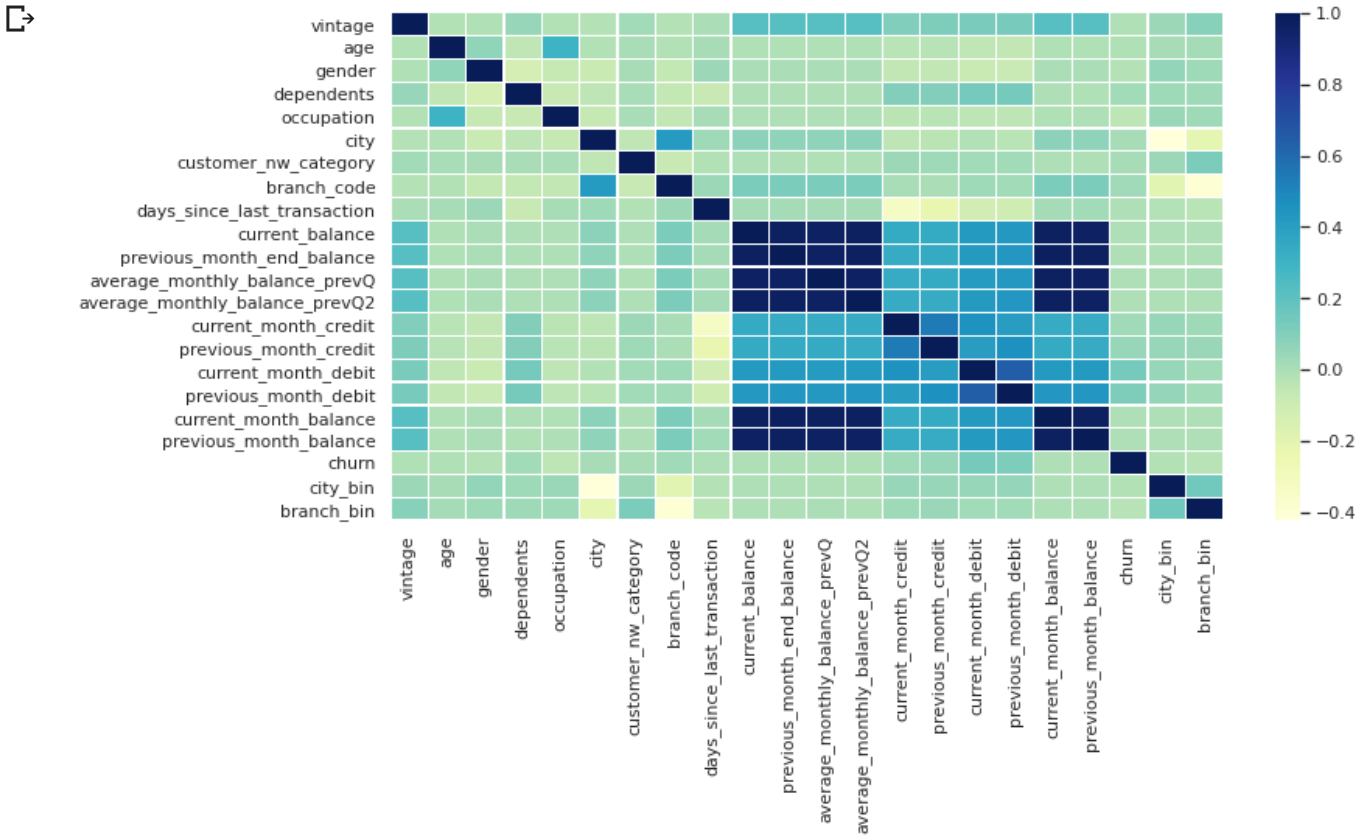
Not much difference in customer net worth category when it comes to churn.

Correlation Heatmap

Lastly, we will look at the correlation heatmap to check what all variables are correlated and to what extent.

```
plt.figure(figsize=(12, 6))
df.drop(['customer_id'],
```

```
        axis=1, inplace=True)
corr = df.apply(lambda x: pd.factorize(x)[0]).corr()
ax = sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns,
                 linewidths=.2, cmap="YlGnBu")
```



- The balance features are highly correlated as can be seen from the plot
- Other variables have correlations but on the lower side
- Debit values have the highest correlation amongst the balance features
- Interestingly vintage has a considerable correlation with all the balance features which actually makes sense since older customers will tend to have higher balance

Conclusions

Average customer Profile

Overall a customer at this bank:

- has no dependents
- has been a customer for last 6 years
- predominantly male
- either self employed or salaried customer

EDA Conclusion for Churn

- From the sample, around 17% customers are churning
- Current balance and average monthly balance values have a left skewed distribution as observed from the histogram
- No significant difference in distributions for average monthly balance and month end balances
    - Bimodal distribution/Double Bell Curve shows that there are 2 different types of customers with 2 brackets of credit and debit. Now, during the modeling phase, these could be considered as a seperate set of customers
- For debit values, we see that there is a significant difference in the distribution for churn and non churn and it might be turn out to be an important feature
- For most frequent vintage values, the churning customers are slightly higher, while for higher values of vintage, we have mostly non churning customers which is in sync with the age variable
- Gender does not look like a very significant variable as the ratio of churned customers and others is very similar
- Self Employed and salaried have higher churn rate and are the most frequently occuring categories.
- Not much difference in customer net worth category when it comes to churn

Missing Values

Before we go on to build the model, we must look for missing values within the dataset as treating the missing values is a necessary step before we fit a logistic regression model on the dataset.

```
pd.isnull(df).sum()
```

```
vintage                       0
age                           0
gender                      525
dependents                 2463
occupation                   80
city                        803
customer_nw_category          0
```

The result of this function shows that there are quite a few missing values in columns gender, dependents, city, days since last transaction and Percentage change in credits. Let us go through each of them 1 by 1 to find the appropriate missing value imputation strategy for each of them.

Gender

For a quick recall let us look at the categories within gender column

```
df['gender'].value_counts()
```

```
Male      16548
Female    11309
Name: gender, dtype: int64
```

So there is a good mix of males and females and arguably missing values cannot be filled with any one of them. We could create a seperate category by assigning the value -1 for all missing values in this column.

Before that, first we will convert the gender into 0/1 and then replace missing values with -1.

```
#Convert Gender
dict_gender = {'Male': 1, 'Female':0}
df.replace({'gender': dict_gender}, inplace = True)

df['gender'] = df['gender'].fillna(-1)
```

Dependents, occupation and city with mode

Next, we will have a quick look at the dependents & occupations column and impute with mode as this is sort of an ordinal variable.

```
df['dependents'].value_counts()
```

```
0.0    21435
2.0     2150
1.0     1395
3.0      939
Name: dependents, dtype: int64
```

```
df['occupation'].value_counts()
```

```
self_employed    17476
salaried          6704
student           2058
retired           2024
company             40
Name: occupation, dtype: int64
```

```
df['dependents'] = df['dependents'].fillna(0)
df['occupation'] = df['occupation'].fillna('self_employed')
```

```
df['city'] = df['city'].fillna(1020)
```

Days since Last Transaction

A fair assumption can be made on this column as this is number of days since last transaction in 1 year, we can substitute missing values with a value greater than 1 year say 999.

```
df['days_since_last_transaction'] = df['days_since_last_transaction'].fillna(999)
```

Preprocessing

Now, before applying linear model such as logistic regression, we need to scale the data and keep all features as numeric strictly.

Dummies with Multiple Categories

```
# Convert occupation to one hot encoded features
df = pd.concat([df,pd.get_dummies(df['occupation'],prefix = str('occupation'),prefix_sep='
```

Scaling Numerical Features

Now, we remember that there are a lot of outliers in the dataset especially when it comes to previous and current balance features. Also, the distributions are skewed for these features if you recall from the EDA. We will take 2 steps to deal with that here:

1) Log Transformation

2) Standard Scaler

Standard scaling is anyways a necessity when it comes to linear models and we have done that here after doing log transformation on all balance features.

```
num_cols = ['customer_nw_category', 'current_balance'
```

```
num_cols = ['customer_nw_category', 'current_balance',
            'previous_month_end_balance', 'average_monthly_balance_prevQ2', 'average_month
            'current_month_credit','previous_month_credit', 'current_month_debit',
            'previous_month_debit','current_month_balance', 'previous_month_balance']
for i in num_cols:
    df[i] = np.log(df[i] + 17000)

std = StandardScaler()
scaled = std.fit_transform(df[num_cols])
scaled = pd.DataFrame(scaled,columns=num_cols)


df_df_og = df.copy()
df = df.drop(columns = num_cols,axis = 1)
df = df.merge(scaled,left_index=True,right_index=True,how = "left")


y_all = df.churn
df = df.drop(['churn','occupation'],axis = 1)
```

Model Building and Evaluation Metrics

Since this is a binary classification problem, we could use the following 2 popular metrics:

1. Recall
2. Area under the Receiver operating characteristic curve

Now, we are looking at the recall value here because a customer falsely marked as churn would not be as bad as a customer who was not detected as a churning customer and appropriate measures were not taken by the bank to stop him/her from churning.

The ROC AUC is the area under the curve when plotting the (normalized) true positive rate (x-axis) and the false positive rate (y-axis).

Our main metric here would be Recall values, while AUC ROC Score would take care of how well predicted probabilites are able to differentiate between the 2 classes.

Conclusions from EDA

- For debit values, we see that there is a significant difference in the distribution for churn and non churn and it might be turn out to be an important feature.
- For all the balance features the lower values have much higher proportion of churning customers.
- For most frequent vintage values, the churning customers are slightly higher, while for higher values of vintage, we have mostly non churning customers which is in sync with the age variable
- We see significant difference for different occupations and certainly would be interesting to use as a feature for prediction of churn.

Now, we will first split our dataset into test and train and using the above conclusions select columns and build a baseline logistic regression model to check the ROC-AUC Score & the confusion matrix

Baseline Columns

```
baseline_cols = ['current_month_debit', 'previous_month_debit','current_balance','previous
                 ,'occupation_retired', 'occupation_salaried','occupation_self_employed',
df_baseline = df[baseline_cols]
```

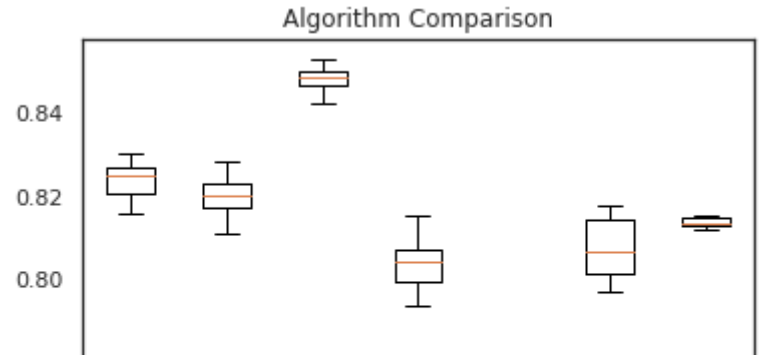
Train Test Split to create a validation set

```
# Splitting the data into Train and Validation set
xtrain, xtest, ytrain, ytest = train_test_split(df_baseline,y_all,test_size=1/3, random_st
```

```
models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('RF', RandomForestClassifier()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))
# evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
    cv_results = cross_val_score(model, xtrain, ytrain, cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))
```

```
LR: 0.824005 (0.004525)
LDA: 0.819988 (0.004564)
RF: 0.848422 (0.003135)
KNN: 0.803499 (0.006066)
CART: 0.769621 (0.007144)
NB: 0.807409 (0.007120)
SVM: 0.813805 (0.001245)
```

```
# Compare Algorithms
plt.boxplot(results, labels=names)
plt.title('Algorithm Comparison')
plt.show()
```




```
rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(xtrain, ytrain)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

```
rfc_pred = rfc.predict(xtest)
```

Classification Report and Confusion Matrix

```
print(confusion_matrix(ytest,rfc_pred))
```

```
[[7352  356]
 [1033  720]]
```

```
print(classification_report(ytest,rfc_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.95      0.91      7708
           1       0.67      0.41      0.51      1753

    accuracy                           0.85      9461
   macro avg       0.77      0.68      0.71      9461
weighted avg       0.84      0.85      0.84      9461
```

```
print( accuracy_score(ytest,rfc_pred))
```

```
0.8531867667265617
```

Hence, We got our desired result by using Random Forest Machine Learning algorithm.