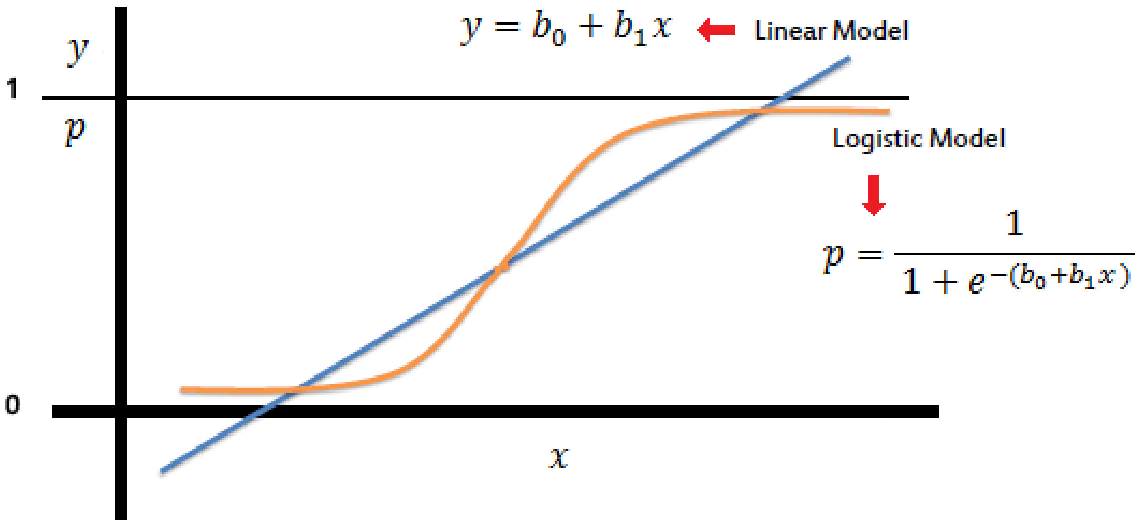
LOGISTIC REGRESSION MODEL



Predicting whether a customer of a certain bank shall churn or not based on multiple parameters using a Logistic Regression Model.

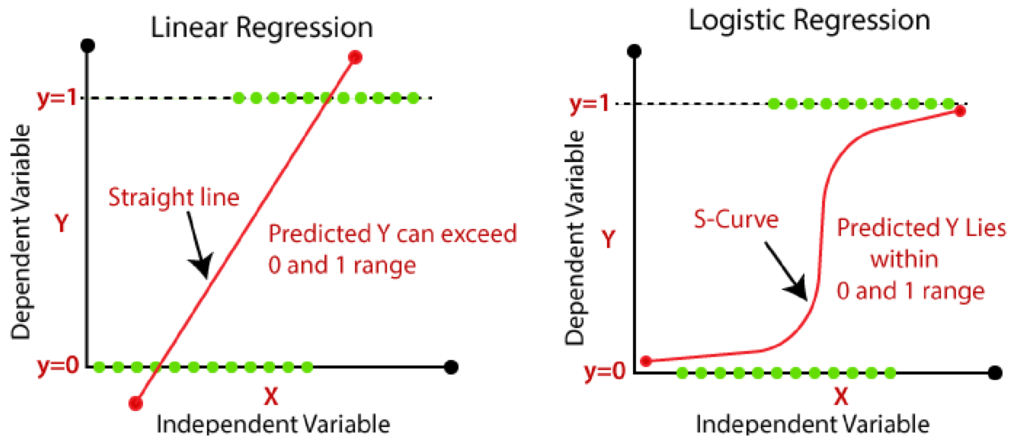
Date	29/03/2020
Name	Vinit Ravichandran Iyer

Content List

Introduction

In statistics, the logistic model (or logit model) is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. This can be extended to model several classes of events such as determining whether an image contains a cat, dog, lion, etc. Each object being detected in the image would be assigned a probability between 0 and 1, with a sum of one. Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable.

Logistic Regression is used when the dependent variable(target) is categorical. For example, To predict whether an email is spam (1) or (0), whether the tumour is malignant (1) or not (0), etc. Linear regression is unbounded, and this brings logistic regression into picture. Their value strictly ranges from 0 to 1.



The output from the hypothesis is the estimated probability. This is used to infer how confident can predicted value be actual value when given an input X. Data is fit into linear regression model, which then be acted upon by a logistic function predicting the target categorical dependent variable. Logistic Regression is of 3 types:

- Binary Logistic Regression
- Multinomial Logistic Regression
- Ordinal Logistic Regression

The following code is of a Binary Logistic Regression which answers the possibility of whether a customer would “churn” or not based on multiple parameters such as the credit score of the current month, past month, current month balance, etc.

Source Code

1. Importing Libraries

Input	import pandas as pd import matplotlib.pyplot as plt import seaborn as sea import numpy as np import warnings warnings.filterwarnings(action = 'ignore')

2. Importing the Dataset

Input	data = pd.read_csv("churn_prediction.csv")
Input	data.head(5)

Output

	customer_id	vintage	age	gender	dependents	occupation	city	customer_nw_category	branch_code	current_balance	...	average_monthly_balance_p
0	1	2101	66	Male	0.0	self_employed	187.0	2	755	1458.71	...	14
1	2	2348	35	Male	0.0	self_employed	NaN	2	3214	5390.37	...	77
2	4	2194	31	Male	0.0	salaried	146.0	2	41	3913.16	...	49
3	5	2329	90	NaN	NaN	self_employed	1020.0	2	582	2291.91	...	20
4	6	1579	42	Male	2.0	self_employed	1494.0	3	388	927.72	...	16

5 rows × 21 columns

3. Studying the Dataset

Input	data.head()

Output

	customer_id	vintage	age	gender	dependents	occupation	city	customer_nw_category	branch_code	current_balance	...	average_monthly_balance_p
0	1	2101	66	Male	0.0	self_employed	187.0	2	755	1458.71	...	14
1	2	2348	35	Male	0.0	self_employed	NaN	2	3214	5390.37	...	77
2	4	2194	31	Male	0.0	salaried	146.0	2	41	3913.16	...	49
3	5	2329	90	NaN	NaN	self_employed	1020.0	2	582	2291.91	...	20
4	6	1579	42	Male	2.0	self_employed	1494.0	3	388	927.72	...	16

5 rows × 21 columns

Input	data.tail()

Output

	customer_id	vintage	age	gender	dependents	occupation	city	customer_nw_category	branch_code	current_balance	...	average_monthly_balar
28377	30297	2325	10	Female	0.0	student	1020.0	2	1207	1076.43	...	
28378	30298	1537	34	Female	0.0	self_employed	1046.0	2	223	3844.10	...	
28379	30299	2376	47	Male	0.0	salaried	1096.0	2	588	65511.97	...	
28380	30300	1745	50	Male	3.0	self_employed	1219.0	3	274	1625.55	...	
28381	30301	1175	18	Male	0.0	student	1232.0	2	474	2107.05	...	

5 rows × 21 columns

Input	data.shape
Output	(28382, 21)

Input	data.columns
Output	Index(['customer_id', 'vintage', 'age', 'gender', 'dependents', 'occupation', 'city', 'customer_nw_category', 'branch_code', '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', 'last_transaction'], dtype='object')

4. Studying the Variables concerned

Input	data.dtypes
Output	customer_id int64 vintage int64 age int64 gender object dependents float64 occupation object city float64 customer_nw_category int64 branch_code int64 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 last_transaction object dtype: object

Input	data.dtypes[data.dtypes == "int64"]
Output	customer_id int64 vintage int64 age int64 customer_nw_category int64 branch_code int64 churn int64 dtype: object

Input	data.dtypes[data.dtypes == "object"]

Output	`gender              object`
	`occupation          object`
	`last_transaction    object`
	`dtype: object`

Input	data.dtypes[data.dtypes == "float64"]

Output	`dependents                        float64`
	`city                              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`
	`dtype: object`

5. Conversion of variables into suitable Data types

Input	data["customer_nw_category"] = data["customer_nw_category"].astype("category") data["branch_code"] = data["branch_code"].astype("category") data["churn"] = data["churn"].astype("category") data["gender"] = data["gender"].astype("category") data["occupation"] = data["occupation"].astype("category") data["city"] = data["city"].astype("category") data["dependents"] = data["dependents"].astype("Int64") data.dtypes

Output	`customer_id                          int64`
	`vintage                              int64`
	`age                                  int64`
	`gender                            category`
	`dependents                           Int64`
	`occupation                        category`
	`city                              category`
	`customer_nw_category              category`
	`branch_code                       category`
	`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                             category`
	`last_transaction                    object`
	`dtype: object`

Input

```
date = pd.DatetimeIndex(data["last_transaction"])

data["doy_lt"] = date.dayofyear #Day of Year of Last Transaction
data["woy_lt"] = date.weekofyear #Week of Year of Last Transaction
data["moy_lt"] = date.month #Month of Yeaar of Last Transaction
data["dow_lt"] = date.dayofweek #Day of Week of Last Transaction

data.head()
```

Output

alance	...	current_month_debit	previous_month_debit	current_month_balance	previous_month_balance	churn	last_transaction	doy_lt	woy_lt	moy_lt	dow_lt
458.71	...	0.20	0.20	1458.71	1458.71	0	2019-05-21	141.0	21.0	5.0	1.0
390.37	...	5486.27	100.56	6496.78	8787.61	0	2019-11-01	305.0	44.0	11.0	4.0
913.16	...	6046.73	259.23	5006.28	5070.14	0	NaT	NaN	NaN	NaN	NaN
291.91	...	0.47	2143.33	2291.91	1669.79	1	2019-08-06	218.0	32.0	8.0	1.0
927.72	...	588.62	1538.06	1157.15	1677.16	1	2019-11-03	307.0	44.0	11.0	6.0

6. Deleting the Excess Columns

Input

```
data = data.drop(columns = ["last_transaction", "Day_of_Year_of_Last_Transaction", "Week_of_Year_of_Last_Transaction", "Month_of_Year_of_Last_Transaction", "Day_of_Week_of_Last_Transaction"])
data.head()

data.dtypes
```

Output

```
customer_id                      int64
vintage                          int64
age                              int64
gender                        category
dependents                       Int64
occupation                    category
city                          category
customer_nw_category          category
branch_code                   category
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                         category
last_transaction                object
doy_lt                         float64
woy_lt                         float64
moy_lt                         float64
dow_lt                         float64
dtype: object
```

7. Univariate Analysis – Numerical values

Input	data.select_dtypes(include = ["int64", "Int64", "float64"]).dtypes
Output	customer_id int64 vintage int64 age int64 dependents Int64 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 doy_lt float64 woy_lt float64 moy_lt float64 dow_lt float64 dtype: object

- **Grouping of Variables**

Input	customer_details=["customer_id", "vintage", "age"] current_month_details=["current_balance", "current_month_credit", "current_month_debit", "current_month_balance"] previous_month_details=["previous_month_end_balance", "previous_month_credit", "previous_month_debit", "previous_month_balance"] previous_quarter_details=["average_monthly_balance_prevQ", "average_monthly_balance_prevQ2"] transaction_date=["doy_lt", "woy_lt", "moy_lt", "dow_lt"]

- **Defining function for Univariate Analysis**

Input

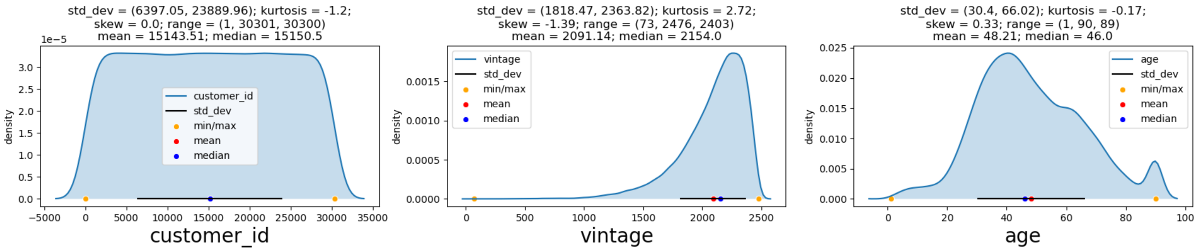
```
def UVA_numeric(data, var_group):
  size = len(var_group)
  plt.figure(figsize = (7*size,3), dpi = 100)

  for j,i in enumerate(var_group):
    mini = data[i].min()
    maxi = data[i].max()
    ran = data[i].max()-data[i].min()
    mean = data[i].mean()
    median = data[i].median()
    st_dev = data[i].std()
    skew = data[i].skew()
    kurt = data[i].kurtosis()

    points = mean-st_dev, mean+st_dev

    plt.subplot(1,size,j+1)
    sea.kdeplot(data[i], shade=True)
```

```
        sea.lineplot(points, [0,0], color = 'black', label = "std_dev")
        sea.scatterplot([mini,maxi], [0,0], color = 'orange', label = "min/max")
        sea.scatterplot([mean], [0], color = 'red', label = "mean")
        sea.scatterplot([median], [0], color = 'blue', label = "median")
        plt.xlabel('{}'.format(i), fontsize = 20)
        plt.ylabel('density')
        plt.title('std_dev = {}; kurtosis = {};\nskew = {}; range = {}\nmean = {}; median
= {}'.format((round(points[0],2),round(points[1],2)),
                                                                round(kurt,2),
                                                                round(skew,2),

(round(mini,2),round(maxi,2),round(ran,2)),
                                                                round(mean,2),
                                                                round(median,2)))
```

- **Customer Details**

Input	UVA_numeric(data,customer_details)

Output



Customer ID: Customer ID is unique to all customers and hence has the uniform distribution Thus Customer ID as a variable can be dropped

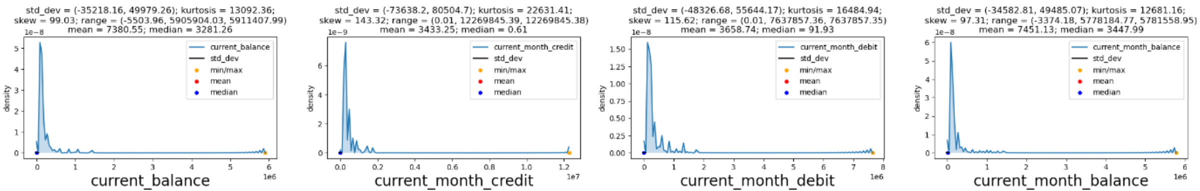
Vintage: It is skewed as many customers joined between 1500 to 2500 days from the day of extraction It has a negative skew therefore variable is associated with the loyal customers Kurtosis is positive and thus outliers may be present

Age: Median age is around 40 Majority of the customers lie between the age of 30-60 It has a positive skew therefore variable is associated to the older customers more Kurtosis is negative and thus outlies to be present is extremely unlikely

- **Current Month**

Input	UVA_numeric(data, current_month_details)

Output

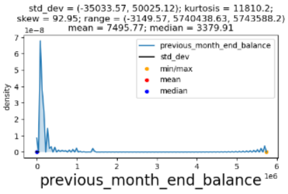


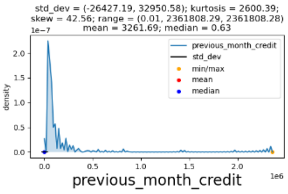
All the above plots have extreme skewness and thus many outliers are present.

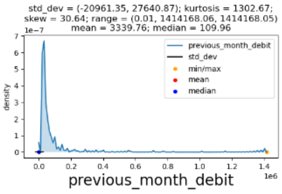
- **Previous Month**

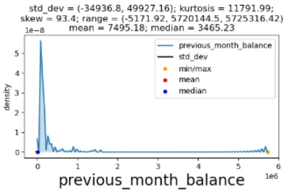
Input	UVA_numeric(data, previous_month_details)

Output







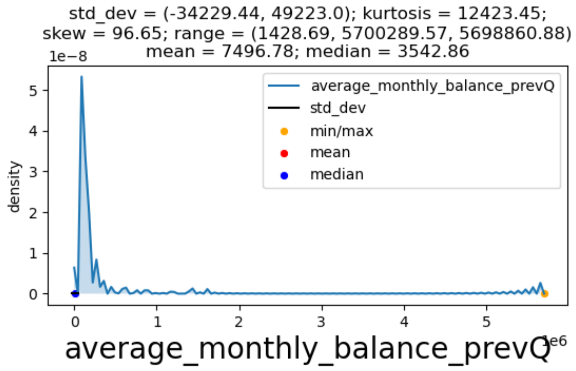


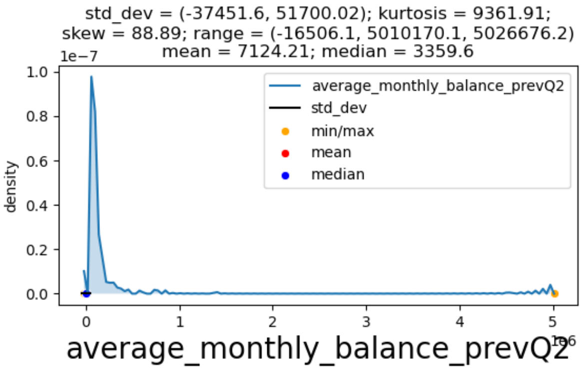
All the above plots have extreme skewness and thus many outliers are present.

- **Previous Quarters**

Input	UVA_numeric(data, previous_quarter_details)

Output



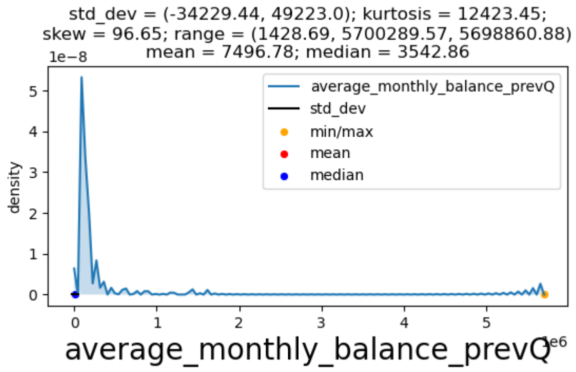


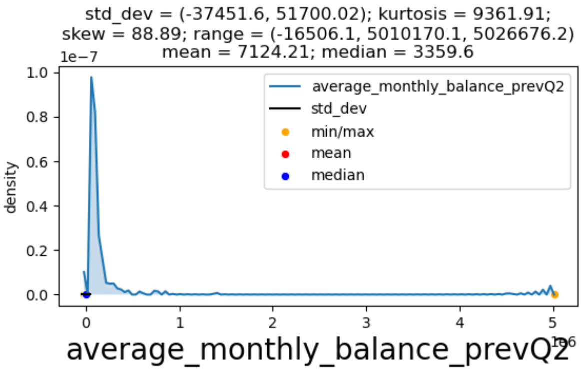
All the above plots have extreme skewness and thus many outliers are present.

- **Transaction Date**

Input	UVA_numeric(data, previous_quarter_details)

Output





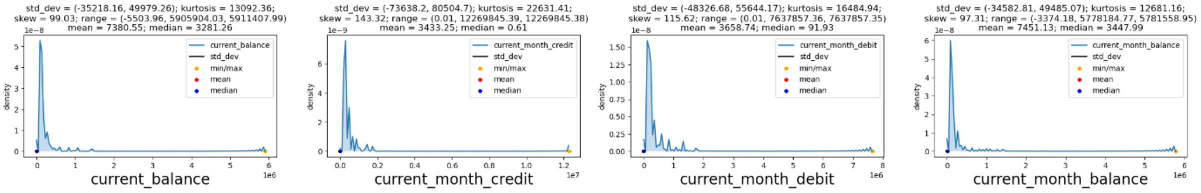
All the above plots have extreme skewness and thus many outliers are present.

- **Removing Outliers for Current month to visualise it**

Input	factor = 3 cm_data = data[current_month_details] cm_data = cm_data[cm_data['current_balance'] < factor*cm_data['current_balance'].std()] cm_data = cm_data[cm_data['current_month_credit'] < factor*cm_data['current_month_credit'].std()] cm_data = cm_data[cm_data['current_month_debit'] < factor*cm_data['current_month_debit'].std()] cm_data = cm_data[cm_data['current_month_balance'] < factor*cm_data['current_month_balance'].std()] len(data), len(cm_data)
Output	(28382, 27113)

Input	UVA_numeric(data, current_month_details)

Output



Even after removal of outliers the skewness is still present. This can hint multiple possibilities such as another relation between them or there might be some customers making high amounts of transactions every month, etc.

8. Univariate Analysis – Categorical Variables

Input	data.select_dtypes(exclude = ["int64", "Int64", "float64"]).dtypes
Output	gender category occupation category city category customer_nw_category category branch_code category churn category last_transaction object dtype: object

Input

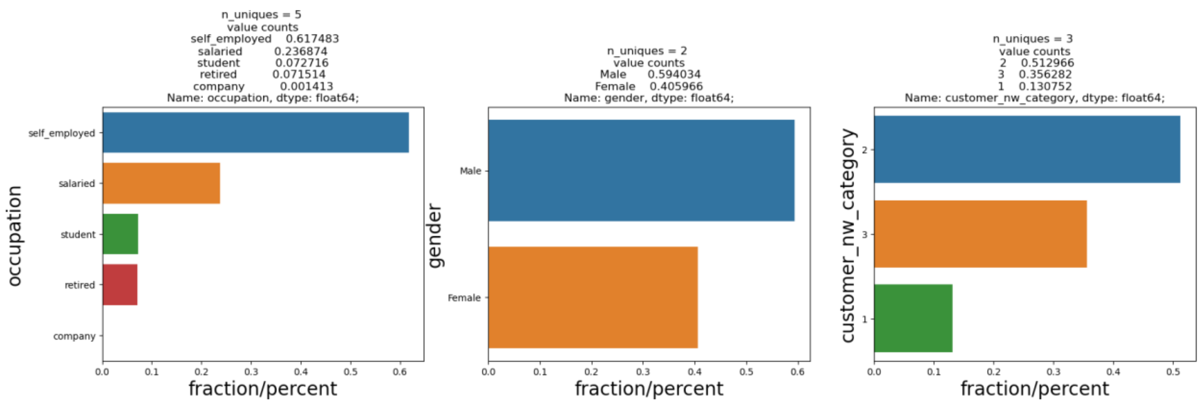
```
def UVA_category(data, var_group):
  size = len(var_group)
  plt.figure(figsize = (7*size,5), dpi = 100)

  for j,i in enumerate(var_group):
    norm_count = data[i].value_counts(normalize = True)
    n_uni = data[i].nunique()

    plt.subplot(1,size,j+1)
    sea.barplot(norm_count, norm_count.index , order = norm_count.index)
    plt.xlabel('fraction/percent', fontsize = 20)
    plt.ylabel('{}'.format(i), fontsize = 20)
    plt.title('n_uniques = {} \n value counts \n {};'.format(n_uni,norm_count))
```

- **Customer Information**

Input

```
UVA_category(data, ['occupation', 'gender', 'customer_nw_category'])
```

Output



We can see that:

Occupation: Majority of the customers are self employed There is little to no company accounts

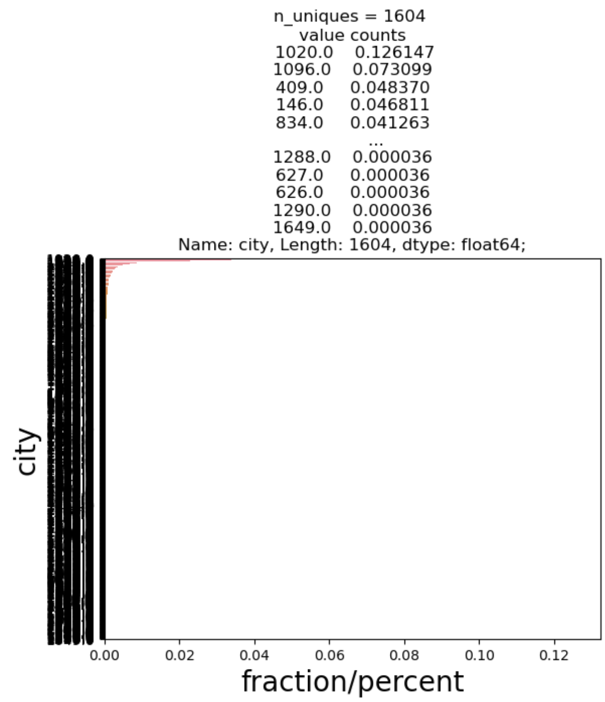
Gender: Males hold more number of accounts than females

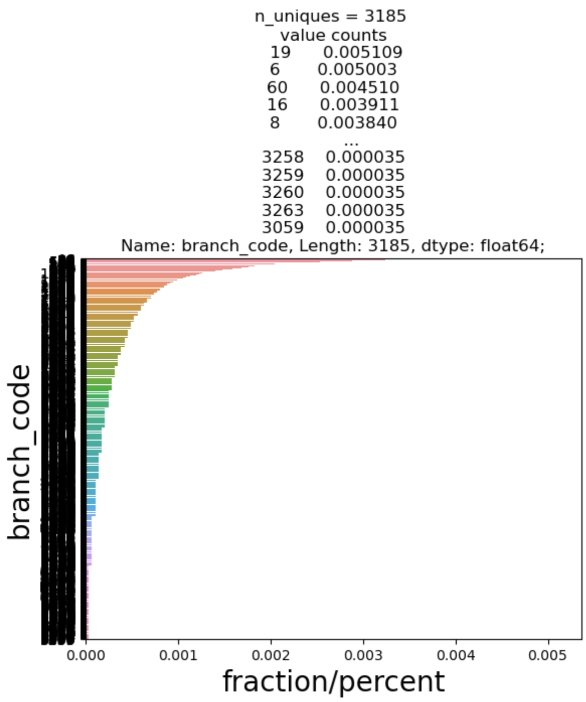
Customer Net Worth: Majority of the customers fall under the 2nd category Median number in category 3 Low number in category 1

- **Account Information**

Input	UVA_category(data, ['city', 'branch_code'])

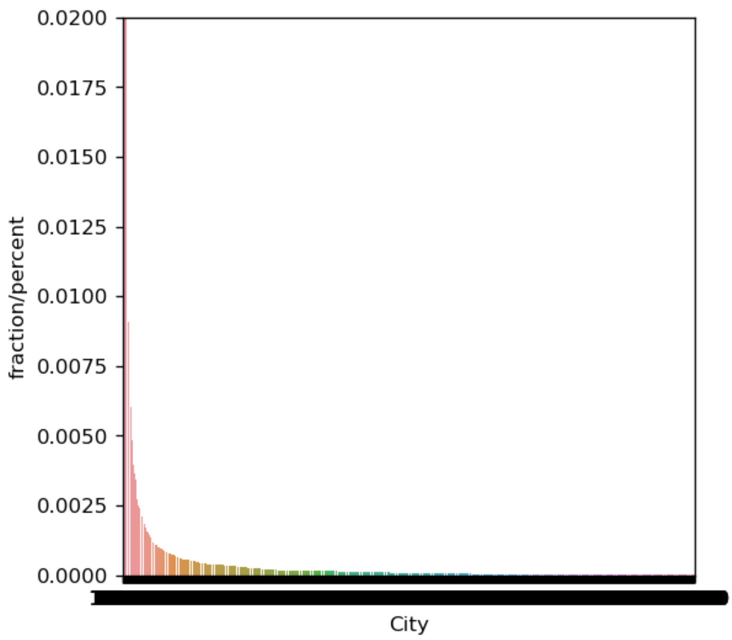
Output





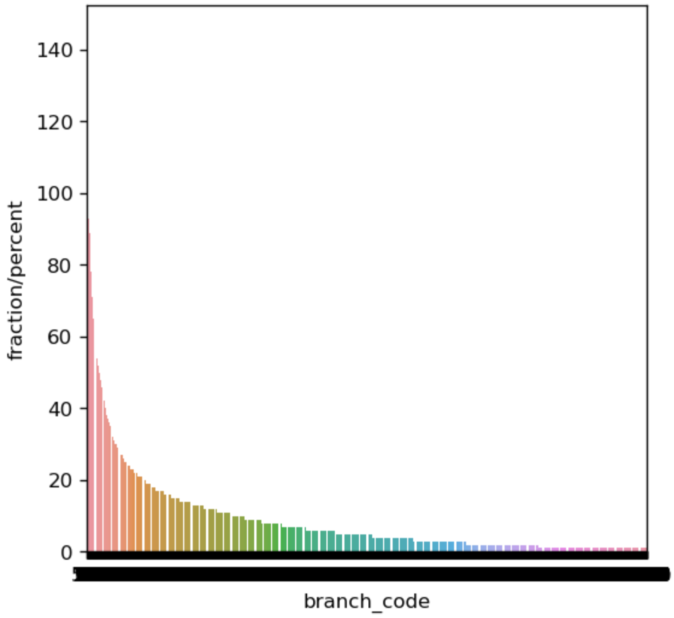
<table>
<tr><td>Input</td><td>#Plotting "city"
plt.figure(figsize = (5,5), dpi = 120)
city_count = data['city'].value_counts(normalize=True)
sea.barplot(city_count.index, city_count , order = city_count.index)
plt.xlabel('City')
plt.ylabel('fraction/percent')
plt.ylim(0,0.02)</td></tr>
</table>

Output



Input	#Plotting "branch_code"
	plt.figure(figsize = (5,5), dpi = 120)
	branch_count = data['branch_code'].value_counts()
	sea.barplot(branch_count.index, branch_count , order = branch_count.index)
	plt.xlabel('branch_code')
	plt.ylabel('fraction/percent')
	#plt.ylim(0,0.02)

Output

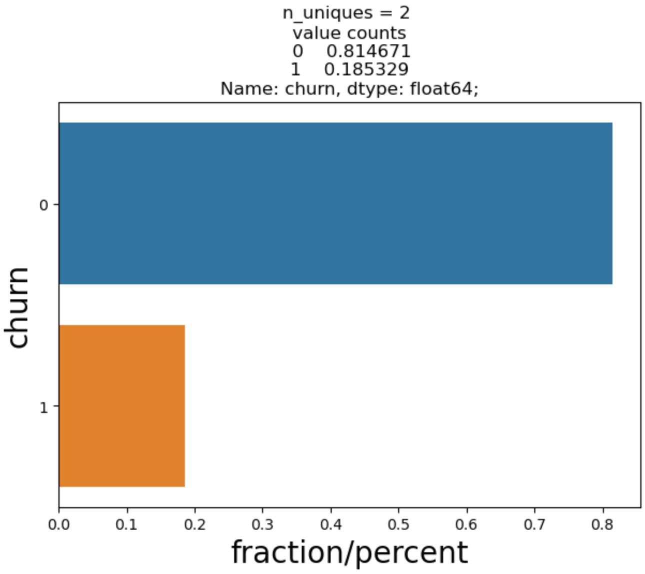


Popular cities and branch code might be able to explain the skewness and outliers of credit/debit variables. Possibility that cities and branch code with very few accounts may lead to churning

- **Churn**

Input	UVA_category(data, ['churn'])

Output



Clearly seen that only 1/4th of the total customers churn

9. Univariate Analysis for Missing Values

Input	data.isnull().sum()

Output	`customer_id 0`
	`vintage 0`
	`age 0`
	`gender 525`
	`dependents 2463`
	`occupation 80`
	`city 803`
	`customer_nw_category 0`
	`branch_code 0`
	`current_balance 0`
	`previous_month_end_balance 0`
	`average_monthly_balance_prevQ 0`
	`average_monthly_balance_prevQ2 0`
	`current_month_credit 0`
	`previous_month_credit 0`
	`current_month_debit 0`
	`previous_month_debit 0`
	`current_month_balance 0`
	`previous_month_balance 0`
	`churn 0`
	`last_transaction 0`
	`doy_lt 3223`
	`woy_lt 3223`
	`moy_lt 3223`
	`dow_lt 3223`
	`dtype: int64`

Things to investigate further down:

- Gender: Do the customers with missing gender values have some common behaviour in-
    - churn: do missing values have any relation with churn?
- Dependents:
    - Missing values might be similar to zero dependents
    - churn: do missing values have any relation with churn?
- Occupation:
    - Do missing values have similar behaviour to any other occupation
    - do they have some relation with churn?
- city:
    - the respective cities can be found using branch_code
- last_transaction:
    - checking their previous month and current month and previous_quarter activity might give insight on their last transaction.
- For almost all the above:
    - vintage: might be recording errors from same period of joining
    - branch_code: might be recording error from certain branch

10. Univariate Analysis for Outliers

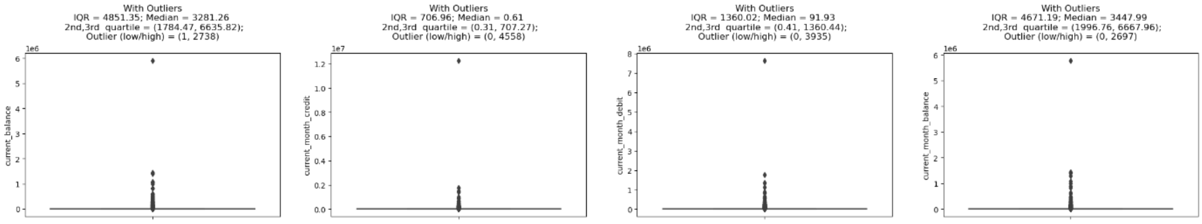
Input	UVA_category(data, ['churn'])

Output	def UVA_outlier(data, var_group, include_outlier = True):
	  size = len(var_group)
	  plt.figure(figsize = (7*size,4), dpi = 100)
	
	  for j,i in enumerate(var_group):
	    quant25 = data[i].quantile(0.25)
	    quant75 = data[i].quantile(0.75)
	    IQR = quant75 - quant25
	    med = data[i].median()
	    whis_low = quant25-(1.5*IQR)
	    whis_high = quant75+(1.5*IQR)
	
	    outlier_high = len(data[i][data[i]>whis_high])
	    outlier_low = len(data[i][data[i]<whis_low])
	
	    if include_outlier == True:
	      plt.subplot(1,size,j+1)
	      sea.boxplot(data[i], orient="v")
	      plt.ylabel('{}'.format(i))
	      plt.title('With Outliers\nIQR = {}; Median = {} \n 2nd,3rd quartile = {};\n Outlier (low/high) = {} \n'.format(
	round(IQR,2),
	round(med,2),
	
	(round(quant25,2),round(quant75,2)),
	(outlier_low,outlier_high)
	))
	    else:
	      data2 = data[var_group][:]
	      data2[i][data2[i]>whis_high] = whis_high+1
	      data2[i][data2[i]<whis_low] = whis_low-1
	
	      plt.subplot(1,size,j+1)
	      sea.boxplot(data2[i], orient="v")
	      plt.ylabel('{}'.format(i))
	      plt.title('Without Outliers\nIQR = {}; Median = {} \n 2nd,3rd quartile = {};\n Outlier (low/high) = {} \n'.format(
	round(IQR,2),
	round(med,2),
	
	(round(quant25,2),round(quant75,2)),
	(outlier_low,outlier_high)
	))

- **Current Month**

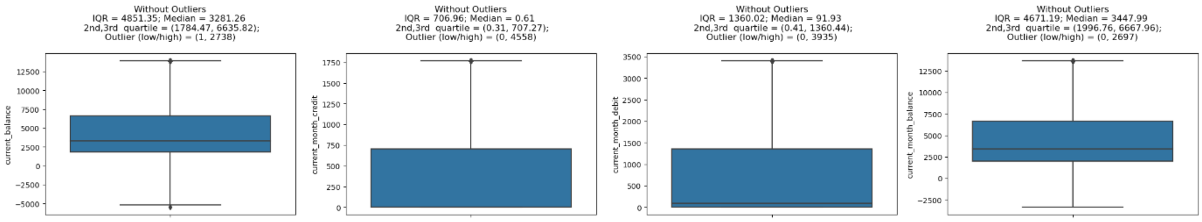
Input	UVA_outlier(data, current_month_details,)

Output



Input	UVA_outlier(data, current_month_details, include_outlier=False)

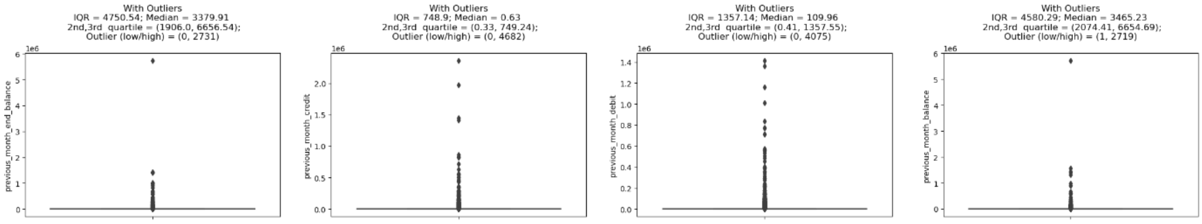
Output



- **Previous Month**

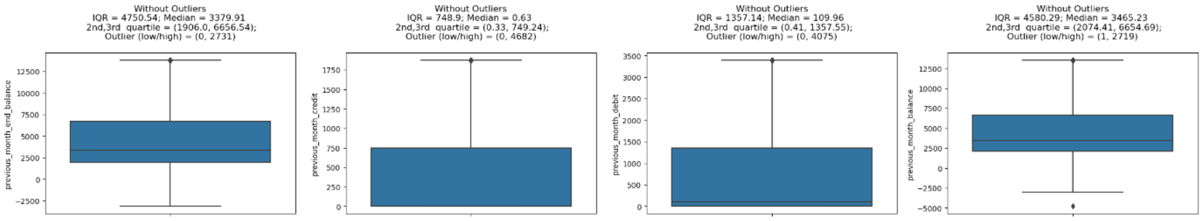
Input	UVA_outlier(data, previous_month_details)

Output



Input	UVA_outlier(data, previous_month_details, include_outlier=False)

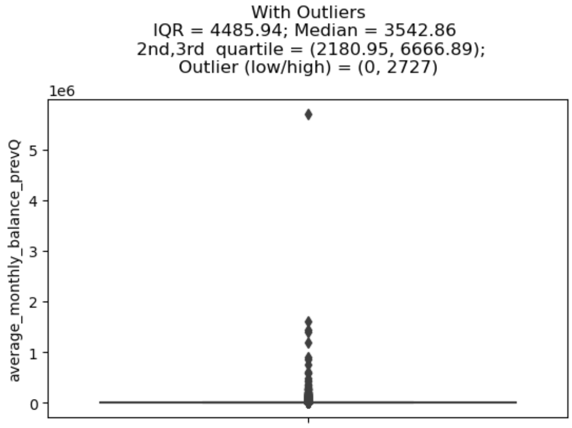
Output

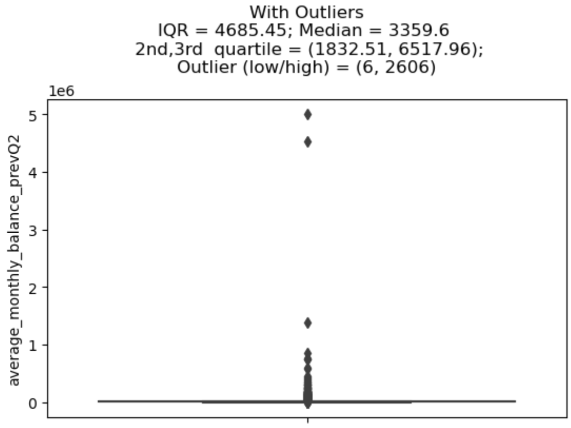


- **Previous Quarters**

Input	UVA_outlier(data,previous_quarter_details)

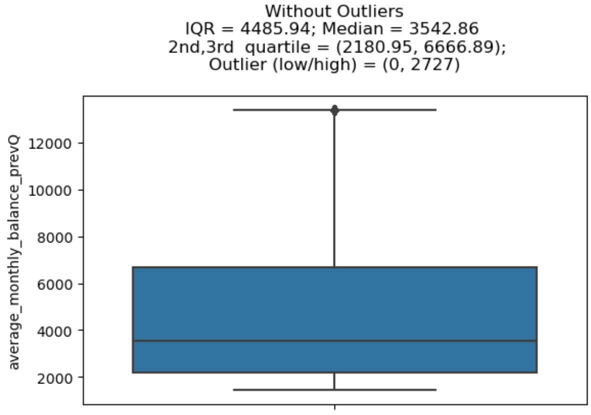
Output

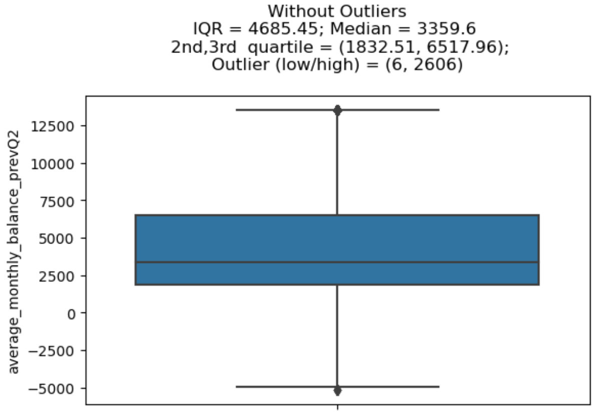




Input	UVA_outlier(data,previous_quarter_details, include_outlier=False)

Output





Investigation directions from Univariate Analysis

1. customer_id variable can be dropped.
2. Is there there any common trait/relation between the customers who are performing high transaction credit/debits? .*customer_nw_category might explain that.* .Occupation = Company might explain them .*popular cities might explain this
3. Customers whose last transaction was 6 months ago, did all of them churn? .*Possibility that cities and branch code with very few accounts may lead to churning.

11. Bivariate Analysis: Numerical-Numerical

Input

```
numerical = data.select_dtypes(include = ["int64",  "Int64", "float64"])
numerical.dtypes
```

Output

```
customer_id                         int64
vintage                             int64
age                                 int64
dependents                          Int64
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
doy_lt                            float64
woy_lt                            float64
moy_lt                            float64
dow_lt                            float64
dtype: object
```

12. Corelation Matrix

Input

```
correlation = numerical.dropna().corr()
correlation
```

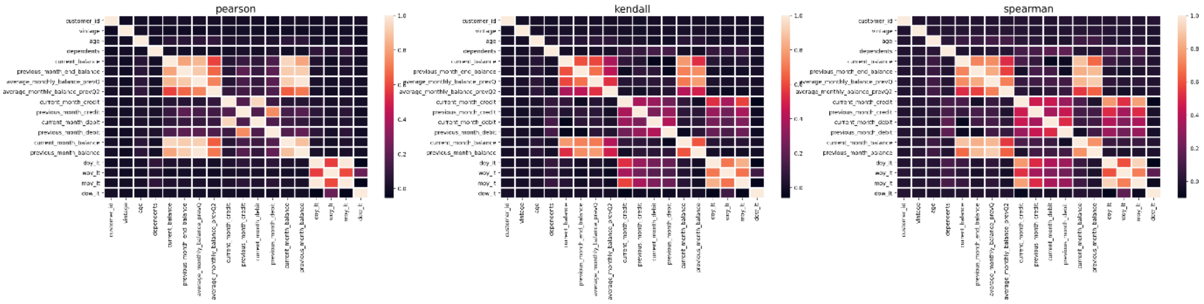
Output

	customer_id	vintage	age	dependents	current_balance	previous_month_end_balance	average_monthly_balance_prev
customer_id	1.000000	-0.011288	0.001397	-0.009737	0.014989	0.012414	0.01137
vintage	-0.011288	1.000000	0.003170	0.005109	-0.007223	-0.008001	-0.01085
age	0.001397	0.003170	1.000000	-0.003809	0.058925	0.062775	0.07090
dependents	-0.009737	0.005109	-0.003809	1.000000	-0.004554	-0.000826	0.00012
current_balance	0.014989	-0.007223	0.058925	-0.004554	1.000000	0.809257	0.85720
previous_month_end_balance	0.012414	-0.008001	0.062775	-0.000826	0.809257	1.000000	0.90805
average_monthly_balance_prevQ	0.011372	-0.010858	0.070903	0.000121	0.857204	0.908053	1.00000
average_monthly_balance_prevQ2	0.008060	-0.003824	0.081361	0.002584	0.584156	0.661439	0.73195
current_month_credit	0.004223	-0.004821	0.023921	0.002188	0.053329	0.051080	0.05129
previous_month_credit	-0.004819	-0.000410	0.027678	0.022772	0.101495	0.195149	0.13896
current_month_debit	0.004870	-0.004899	0.025366	0.006784	0.075149	0.100379	0.09149
previous_month_debit	-0.005906	-0.007777	0.027717	0.029073	0.151771	0.192376	0.18722
current_month_balance	0.012085	-0.008703	0.063120	-0.001859	0.940234	0.910206	0.92094
previous_month_balance	0.011025	-0.010439	0.067712	0.000241	0.812295	0.912269	0.98379
doy_lt	-0.006114	-0.000680	0.010754	0.079740	0.035242	0.024130	0.02110
woy_lt	0.011344	-0.010040	0.000501	0.034460	-0.008980	0.000946	-0.00057
moy_lt	-0.005374	-0.001359	0.011970	0.077978	0.033127	0.023485	0.02094
dow_lt	0.009665	-0.009683	-0.020895	-0.001702	-0.000315	0.002033	0.00064

- **Heatmap for better visualization**

Input	plt.figure(figsize=(36,6), dpi=140)
	for j,i in enumerate(['pearson','kendall','spearman']):
	   plt.subplot(1,3,j+1)
	   correlation = numerical.dropna().corr(method=i)
	   sea.heatmap(correlation, linewidth = 2)
	   plt.title(i, fontsize=18)

Output



1. Kendall and Spearman correlation seem to have very similar pattern between them, except the slight variation in magnitude of correlation.
2. Too many variables with insignificant correlation.
3. Major correlation lies between the transaction variables and balance variables.

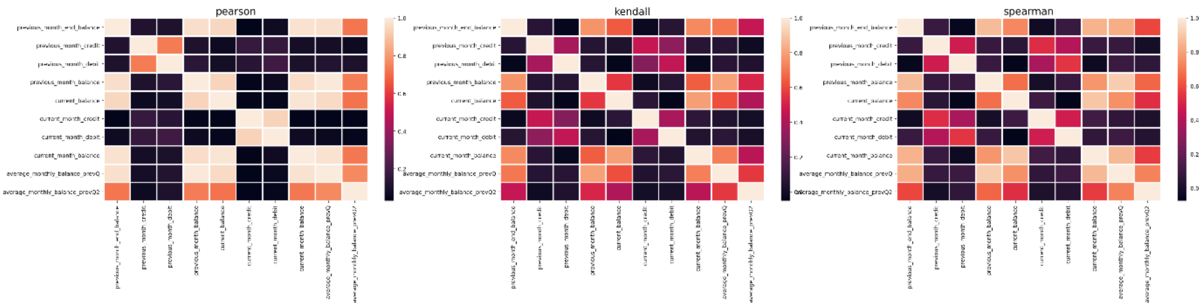
Compiling all the data gathered until now....

Input	var = [] var.extend(previous_month_details) var.extend(current_month_details) var.extend(previous_quarter_details)

Plotting a heatmap with the above included

Input	plt.figure(figsize=(36,6), dpi=140)
	for j,i in enumerate(['pearson','kendall','spearman']):
	   plt.subplot(1,3,j+1)
	   correlation = numerical[var].dropna().corr(method=i)
	   sea.heatmap(correlation, linewidth = 2)
	   plt.title(i, fontsize=18)

Output

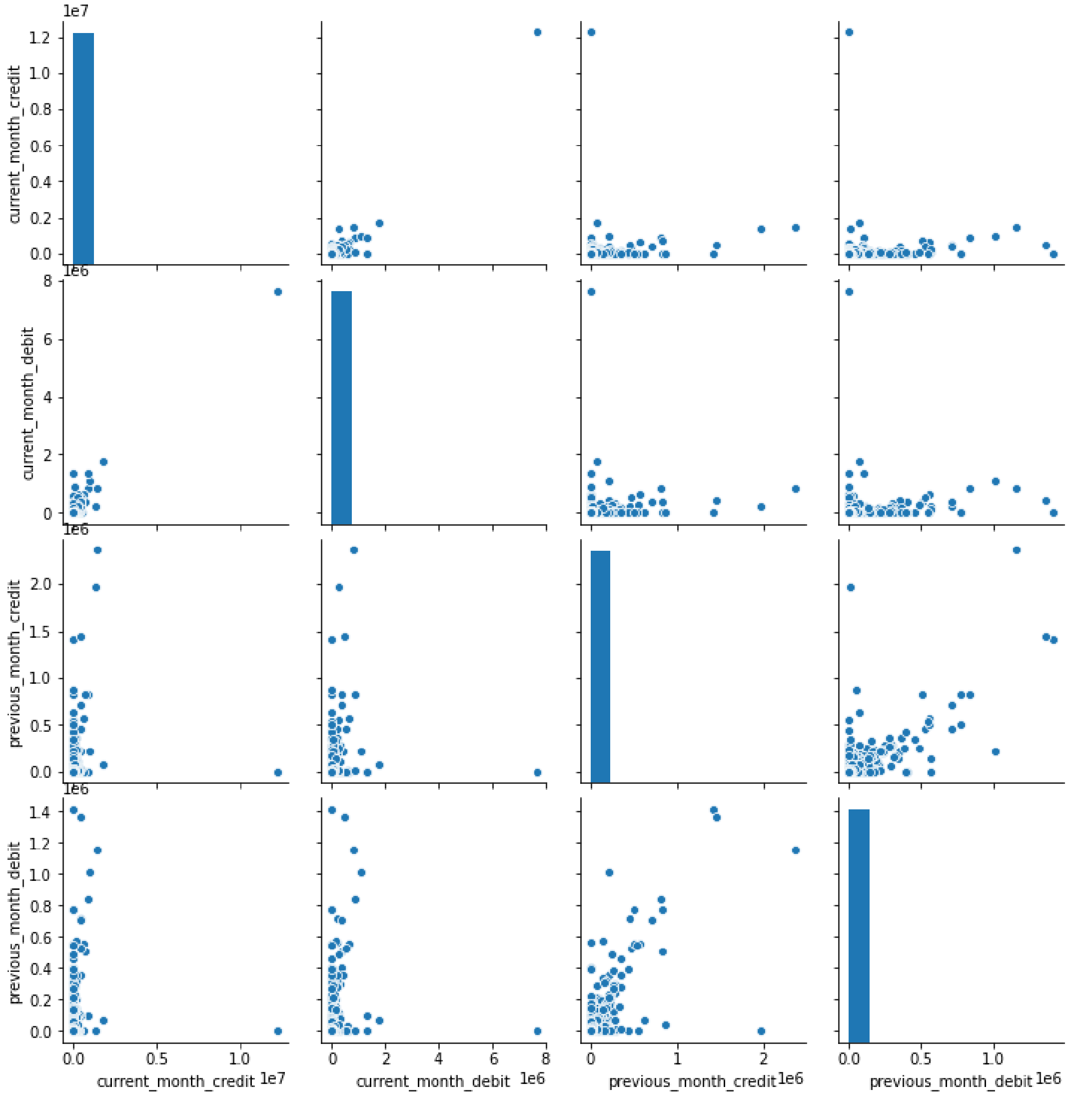


1. Transaction variables like credit/debit have a strong correlation among themselves.
2. Balance variables have strong correlation among themselves.
3. Transaction variables like credit/debit have insignificant or no correlation with the Balance variables.

13. Plotting the Inferences

Input	transactions = ['current_month_credit','current_month_debit','previous_month_credit','previous_month_debit'] balance = ['previous_month_end_balance','previous_month_balance','current_balance','current_month_balance'] plt.figure(dpi=140) sea.pairplot(numerical[transactions])

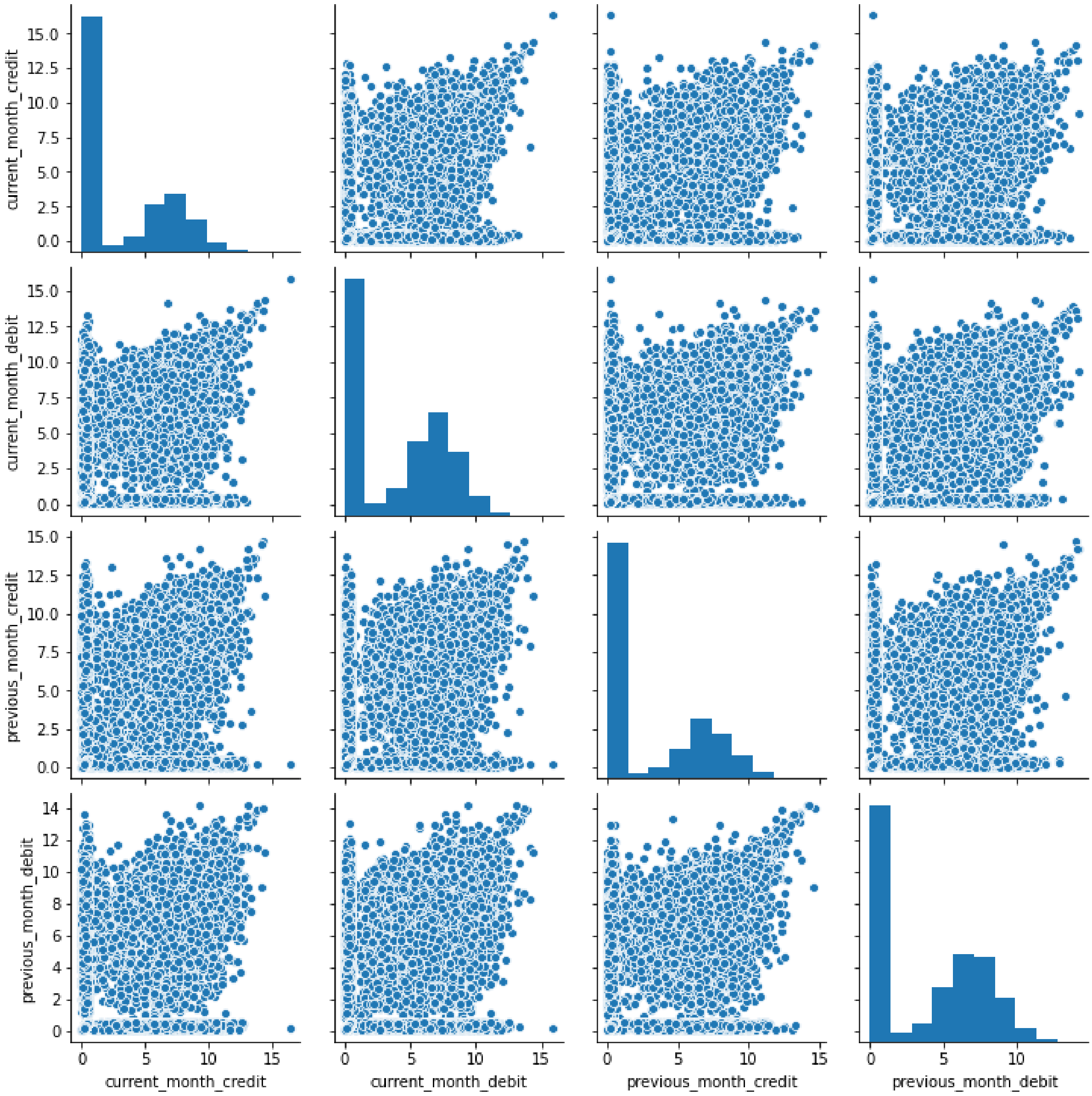
Output



Outliers are present affecting our judgement of the graphs and hence using the log function to negate the effect of the outliers.

Input	for column in var:
	  mini=1
	  if numerical[column].min()<0:
	    mini = abs(numerical[column].min()) + 1
	
	  numerical[column] = [i+mini for i in numerical[column]]
	  numerical[column] = numerical[column].map(lambda x : np.log(x))
	
	plt.figure(dpi=140)
	sea.pairplot(numerical[transactions])

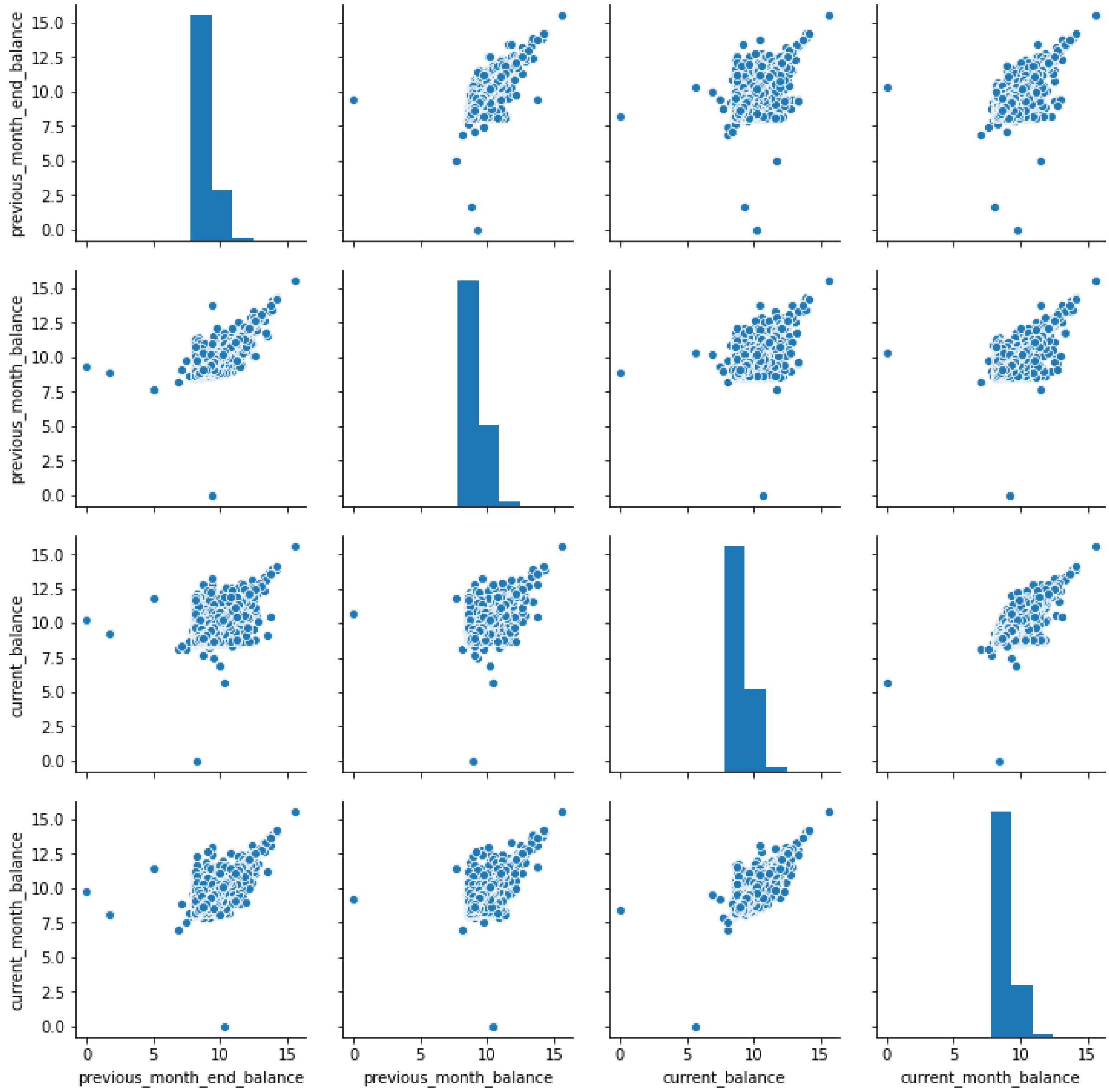
Output



1. This validates the high correlation between the transaction variables.
2. This high correlation can be used for feature engineering during the later stages.

Input	plt.figure(dpi=140) sea.pairplot(numerical[balance])

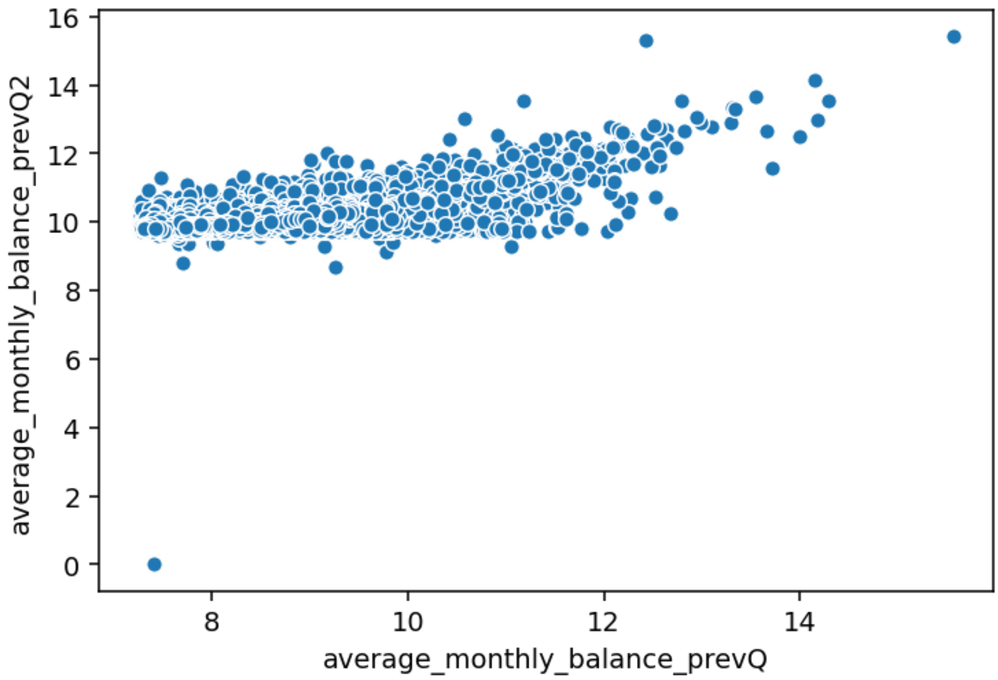
Output



1. This validates the high correlation between the balance variables.
2. This high correlation can be used for feature engineering during the later stages.

Input	plt.figure(dpi=140) sea.scatterplot(numerical['average_monthly_balance_prevQ'], numerical['average_monthly_balance_prevQ2'])

Output



1. This validates the high correlation between the two previous quarters
2. This high correlation can be used for feature engineering during the later stages.

14. Multivariate Analysis

- **Pivot Table – Gender, Occupation, & Customer Net worth category against Churn**

Input	data.dtypes
Output	customer_id int64 vintage int64 age int64 gender category dependents Int64 occupation category city category customer_nw_category category branch_code category 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 category last_transaction object doy_lt float64 woy_lt float64

```
moy_lt                                          float64
dow_lt                                          float64
dtype: object
```

Input

```
data["gender"] = data["gender"].astype("object")
data["occupation"] = data["occupation"].astype("object")
data["customer_nw_category"] = data["customer_nw_category"].astype("object")
data["churn"] = data["churn"].astype("int")
data["city"] = data["city"].astype("float")
data["branch_code"] = data["branch_code"].astype("float")

data.pivot_table("churn", ["gender", "occupation"], "customer_nw_category", aggfunc = "mean")*100
```

Output

	customer_nw_category	1	2	3
gender	**occupation**			
Female	**company**	100.000000	0.000000	66.666667
	retired	20.689655	11.219512	13.492063
	salaried	18.545455	14.849188	17.689016
	self_employed	18.111588	18.197035	18.920916
	student	10.404624	14.442413	15.034965
Male	**company**	0.000000	0.000000	0.000000
	retired	18.497110	14.251781	16.316640
	salaried	17.557252	16.410469	18.468702
	self_employed	22.832370	20.424978	21.465808
	student	16.969697	17.210145	18.208955

1. Highest number of churning customers are those Male Customers who lie in 2 net worth category and belong to Self employed profession
2. Proportion wise for net worth category 1, Approximately 22% Male customers who belong to the Self-employed profession are churning
3. Proportion wise for net worth category 2, 20% Male customers who belong to the Self-employed profession are churning
4. For net worth category 3, Approximately 21% Male customers who belong to the Self-employed profession are churning
5. In all the cases of Customer net worth category, Self-employed Male customers are more likely to churn

- **Pivot Table – Gender, Age & Occupation against Churn**

Input

```
age = pd.cut(data['age'], [0, 25, 50, 100])
data.pivot_table('churn', ['gender', age], 'occupation', aggfunc='mean')*100
```

Output

	occupation	company	retired	salaried	self_employed	student
gender	**age**					
Female	**(0, 25]**	NaN	NaN	15.909091	21.774194	13.421053
	(25, 50]	50.0	0.000000	16.096866	19.163293	15.510204
	(50, 100]	50.0	13.541667	17.948718	17.370083	0.000000
Male	**(0, 25]**	0.0	NaN	20.987654	30.327869	16.545894
	(25, 50]	0.0	14.285714	17.349769	21.886121	21.076233
	(50, 100]	0.0	15.493827	17.165150	19.340538	0.000000

1. We have created three bins for the age variable dividing age into 3 groups 0-25, 25-50 and 50-100
2. Highest number of Customers are churning from Male category who belong to the age group of (25,50) and are professionally self employed
3. Highest Proportion of Customers are churning from Male category who belong to the age group of (0,25) and are professionally self employed
4. Here also Self Employed Male customers are churning more than any other combination of categories

- **Pivot Table – Gender, Age, Occupation and Current Balance against Churn**

Input

```
balance = pd.qcut(data['current_balance'], 3)
data.pivot_table('churn',        ['gender',        age],        [balance,        'occupation'],
aggfunc='mean')*100
```

Output

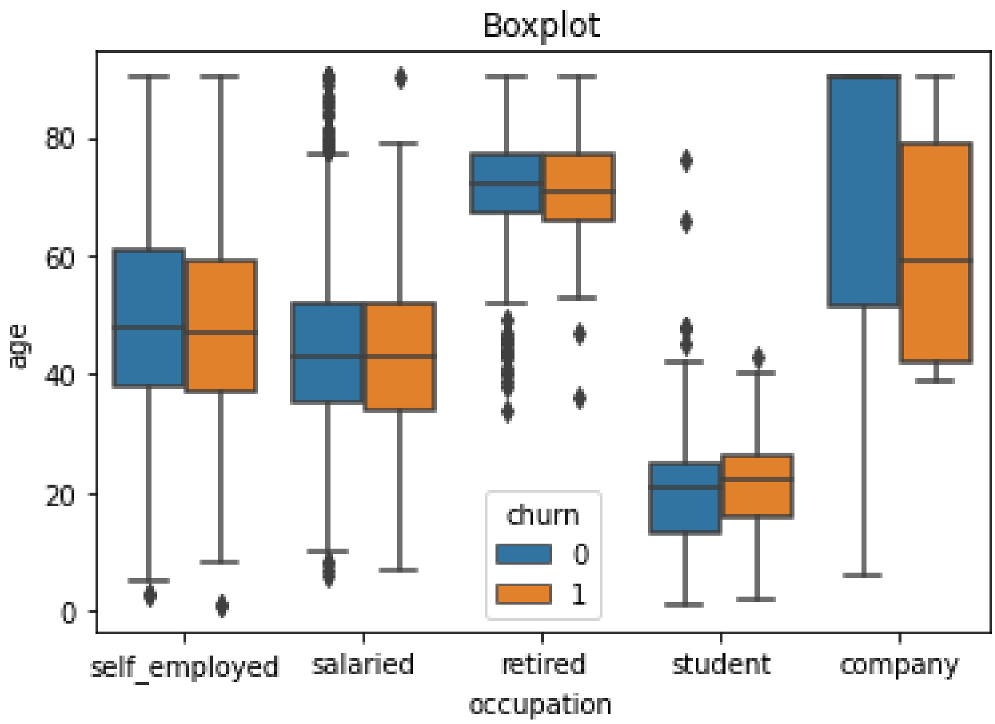
	current_balance	(-5503.961, 2202.177]					(2202.177, 5114.317]					(5114.317, 5905	
	occupation	company	retired	salaried	self_employed	student	company	retired	salaried	self_employed	student	company	retir
gender	**age**												
Female	**(0, 25]**	NaN	NaN	26.315789	38.596491	21.262458	NaN	NaN	5.882353	10.810811	7.167235	NaN	
	(25, 50]	50.0	0.000000	32.300885	33.677419	25.974026	100.0	0.000000	9.826590	10.891720	6.862745	0.0	
	(50, 100]	100.0	28.333333	35.156250	30.642361	0.000000	NaN	5.633803	11.200000	11.052166	NaN	0.0	8.1
Male	**(0, 25]**	0.0	NaN	35.294118	52.000000	28.189911	NaN	NaN	14.285714	14.117647	6.493506	NaN	
	(25, 50]	0.0	0.000000	33.367243	38.901345	44.117647	0.0	16.666667	11.889401	13.214740	12.345679	0.0	20.0
	(50, 100]	0.0	29.489603	32.119914	33.060854	NaN	0.0	6.927176	10.766046	12.565905	NaN	0.0	10.6

1. Current balance is divided into 3 quantiles
2. It is visible at first look that for low current balance more number of customers are churning
3. For the first quantile of current balance, More than 18% (overall average churning) of customers are churning and for second and third quantile percentage of churning customers is less than 18%
4. In first quantile of current balance, for self employed profession as the age increases for customers, their churning proportion decreases. This means that Young Self employed Customers are more prone to churn
5. There is a visible gap in proportion of Self employed females who lie in the age group of (0,25) and Self employed Males who lie in the same group. Young Male Self employed customers are churning more than young female self employed customers

- **Visualising – Age, Occupation and Churn**

Input	def Grouped_Box_Plot(data, cont, cat1, cat2):
	    sea.boxplot(x=cat1, y=cont, hue=cat2, data=data, orient='v')
	    plt.title('Boxplot')
	
	Grouped_Box_Plot(data,'age', 'occupation', 'churn')

Output

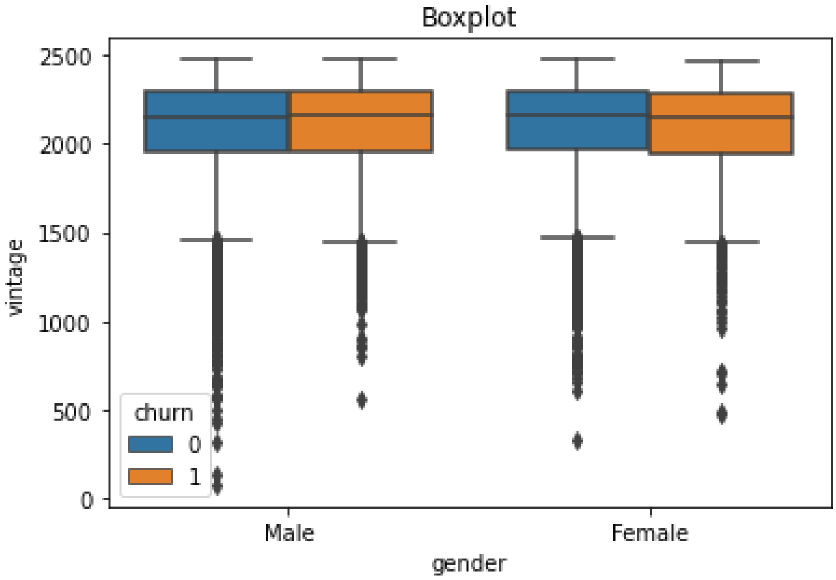


1. For Self-employed profession churning customers are slightly younger than non churning customers
2. In the retired occupation for non churning customers, there are many outliers that indicate young people who retire early are not churning

- **Visualizing – Vintage, Gender & Churn**

Input	Grouped_Box_Plot(data,'vintage','gender', 'churn')

Output



1. There is no visible difference in the vintage feature for gender-wise churning and non churning customers

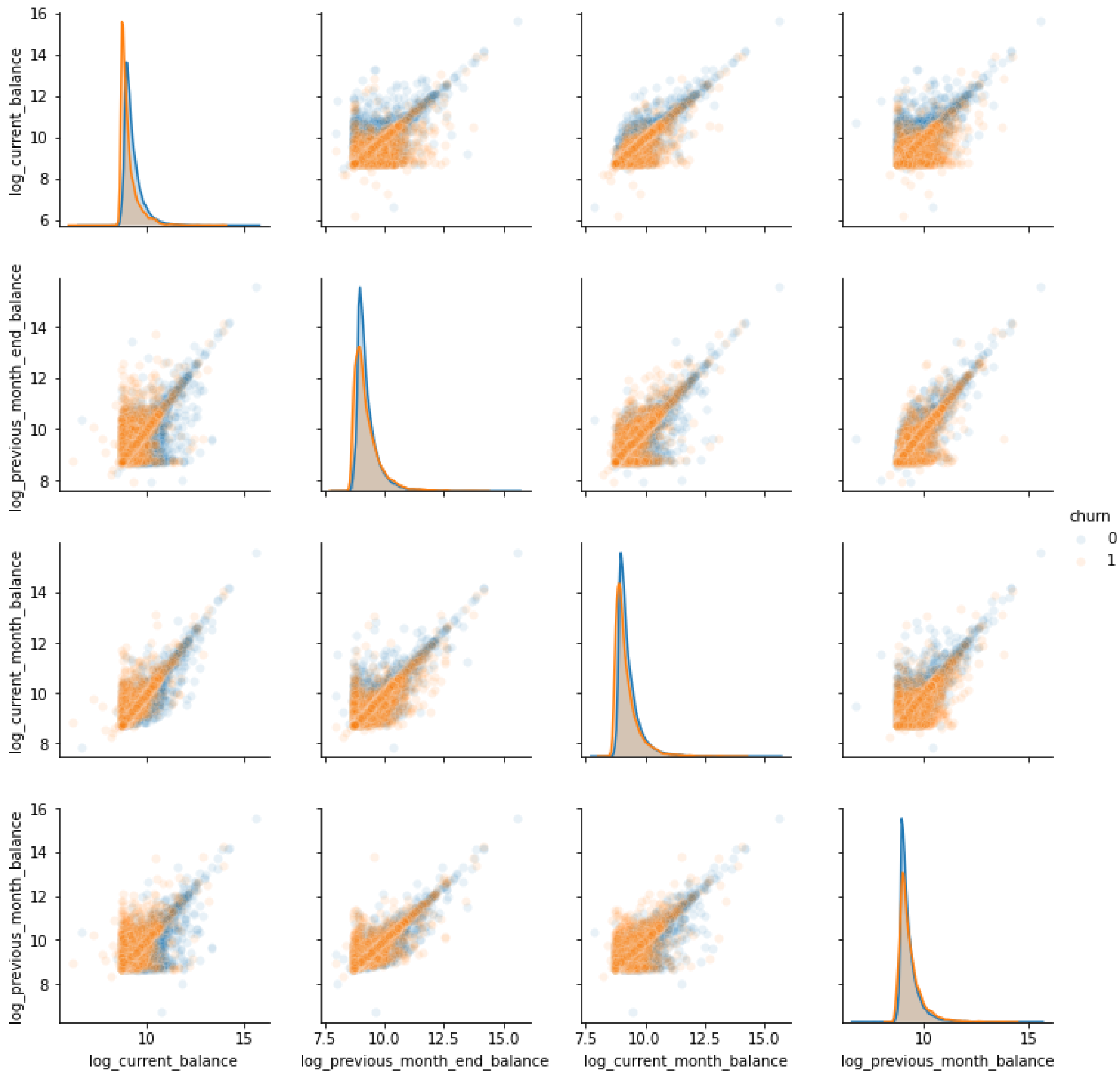
15. Visualizing the Inferences up till now

- **Churn vs Current & Previous Month Balance**

Input

```
balance_cols = ['current_balance','previous_month_end_balance',
            'current_month_balance', 'previous_month_balance']
data1 = pd.DataFrame()

for i in balance_cols:
   data1[str('log_')+ i] = np.log(data[i] + 6000)

log_balance_cols = data1.columns

data1['churn'] = data['churn']

sea.pairplot(data1,vars=log_balance_cols,hue ='churn',plot_kws={'alpha':0.1})
plt.show()
```

Output

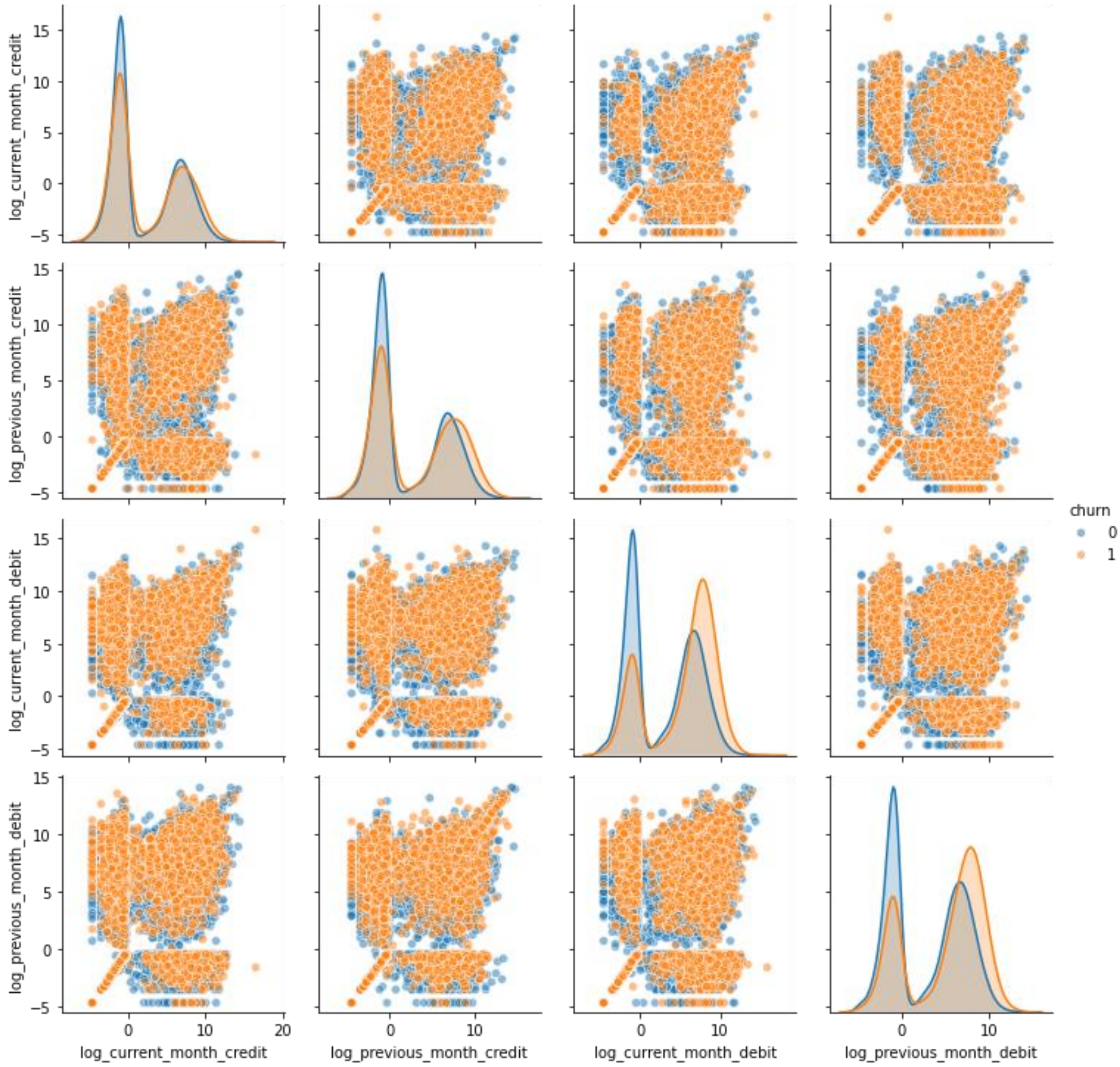


1. There is high correlation between the previous and current month balances which is expected
2. The distribution for churn and not churn is slightly different for both the cases

- **Credit & Debit for Current & Previous Months**

Input	cr_dr_cols = ['current_month_credit','previous_month_credit',
	            'current_month_debit', 'previous_month_debit']
	data1 = pd.DataFrame()
	for i in cr_dr_cols:
	   data1[str('log_')+ i] = np.log(data[i])
	log_dr_cr_cols = data1.columns
	
	data1['churn'] = data['churn']
	sea.pairplot(data1,vars=log_dr_cr_cols, hue = 'churn',plot_kws={'alpha':0.5})
	plt.show()

Output



1. The plots shows that there are 2 different types of customers with 2 brackets of credit and debit.
2. For debit values, we see that there is a significant difference in the distribution for churn and non-churn.

16. Preparing the data for Modelling

- **To make sure every variable has a corresponding numerical value**

Input	data_encoded = pd.get_dummies(data, drop_first=True) data_encoded.head()

Output

	customer_id	vintage	age	dependents	city	branch_code	current_balance	previous_month_end_balance	average_monthly_balance_prevQ	average_montl
0	1	2101	66	0	187.0	755.0	1458.71	1458.71	1458.71	
1	2	2348	35	0	NaN	3214.0	5390.37	8704.66	7799.26	
2	4	2194	31	0	146.0	41.0	3913.16	5815.29	4910.17	
3	5	2329	90	<NA>	1020.0	582.0	2291.91	2291.91	2084.54	
4	6	1579	42	2	1494.0	388.0	927.72	1401.72	1643.31	

5 rows × 388 columns

- **Replacing Missing values with modal numbers**

Input	def fill_mode(df):
	   for column in df.columns:
	      df[column].fillna(df[column].mode()[0], inplace=True)
	
	fill_mode(data_encoded)

- **Splitting data into dependent and independent variables**

Input	data_encoded = data_encoded.drop('customer_id', axis=1) x = data_encoded.drop(['churn'], axis=1) y = data_encoded['churn'] x.shape, y.shape
Ouput	((28382, 386), (28382,))
Input	data_encoded.columns
Output	Index(['vintage', 'age', 'dependents', 'city', 'branch_code', 'current_balance', 'previous_month_end_balance', 'average_monthly_balance_prevQ', 'average_monthly_bala nce_prevQ2', 'current_month_credit', ... 'last_transaction_2019-12-23', 'last_transaction_2019- 12-24', 'last_transaction_2019-12-25', 'last_transaction_2019- 12-26', 'last_transaction_2019-12-27', 'last_transaction_2019- 12-28', 'last_transaction_2019-12-29', 'last_transaction_2019- 12-30', 'last_transaction_2019-12-31', 'last_transaction_NaT'] , dtype='object', length=387)

- **Splitting the data into Training and Test data sets**

Input	from sklearn.model_selection import train_test_split train_x,test_x,train_y,test_y = train_test_split(x,y, random_state = 56)

- **Normalizing the Data**

Input	from sklearn.preprocessing import MinMaxScaler scaler = MinMaxScaler() cols = train_x.columns cols
Output	Index(['vintage', 'age', 'dependents', 'city', 'branch_code', 'current_balance', 'previous_month_end_balance', 'average_monthly_balance_prevQ', 'average_monthly_balance_prevQ2', 'current_month_credit', ... 'last_transaction_2019-12-23', 'last_transaction_2019-12-24', 'last_transaction_2019-12-25', 'last_transaction_2019-12-26', 'last_transaction_2019-12-27', 'last_transaction_2019-12-28', 'last_transaction_2019-12-29', 'last_transaction_2019-12-30', 'last_transaction_2019-12-31', 'last_transaction_NaT'], dtype='object', length=386)
Input	train_x_scaled = scaler.fit_transform(train_x) train_x_scaled = pd.DataFrame(train_x_scaled, columns=cols) train_x_scaled.head()

Output

	vintage	age	dependents	city	branch_code	current_balance	previous_month_end_balance	average_monthly_balance_prevQ	average_monthly_b
0	0.928007	0.348315	0.000000	0.378034	0.499686	0.002316	0.001973	0.000966	
1	0.929255	0.516854	0.000000	0.009102	0.199749	0.000937	0.000563	0.000011	
2	0.956305	0.808989	0.000000	0.618932	0.232796	0.001390	0.000922	0.000114	
3	0.642530	0.258427	0.000000	0.906553	0.210625	0.001194	0.000870	0.000132	
4	0.897628	0.494382	0.019231	0.665049	0.019034	0.010039	0.011464	0.011463	

5 rows × 386 columns

Input	test_x_scaled = scaler.transform(test_x) test_x_scaled = pd.DataFrame(test_x_scaled, columns=cols) test_x_scaled.head()

Output

	vintage	age	dependents	city	branch_code	current_balance	previous_month_end_balance	average_monthly_balance_prevQ	average_monthly_b
0	0.933000	0.516854	0.019231	0.747573	0.021334	0.001343	0.000952	0.000220	
1	0.841448	0.337079	0.000000	0.665049	0.018406	0.001781	0.001474	0.000652	
2	0.917603	0.629213	0.000000	0.374393	0.047689	0.001189	0.000873	0.000020	
3	0.712859	0.269663	0.000000	0.618932	0.230914	0.000911	0.000578	0.000902	
4	0.927591	0.853933	0.000000	0.248180	0.068605	0.000986	0.000654	0.000241	

5 rows × 386 columns

THE DATA TO BE USED FOR MODELLING IS READY

Code Output

<table>
<tr><td>Input</td><td>

```
from sklearn.linear_model import LogisticRegression as LogReg
from sklearn.metrics import accuracy_score
logreg = LogReg()

logreg.fit(train_x, train_y)

train_predict = logreg.predict(train_x)
train_predict

k = accuracy_score(train_predict, train_y)
print('Training accuracy_score', k )

test_predict = logreg.predict(test_x)
k = accuracy_score(test_predict, test_y)
print('Test accuracy_score    ', k )
```

</td></tr>
<tr><td>Output</td><td>

```
Training accuracy_score 0.8235929719064173
Test accuracy_score     0.8275084554678692
```

</td></tr>
</table>

Inference

As can be seen from the Code Output, the variables encoded when normalizing the data does affect the probability whether a customer would churn or not. Churn is expressed as a degree of customer inactivity or disengagement, observed over a given time. Based on the recency of activity which can be viewed from the encoded variables one can predict by an accuracy of 82.75% whether the customer of the bank will churn or not.