Problem statement

SuperLender is a local digital lending company in Nigeria, which prides itself in its effective use of credit risk models to deliver profitable and high-impact loan alternative. Its assessment approach is based on two main risk drivers of loan default prediction:. 1) willingness to pay and 2) ability to pay. Since not all customers pay back, the company invests in experienced data scientist to build robust models to effectively predict the odds of repayment.

These two fundamental drivers need to be determined at the point of each application to allow the credit grantor to make a calculated decision based on repayment odds, which in turn determines if an applicant should get a loan, and if so - what the size, price and tenure of the offer will be.

There are two types of risk models in general: New business risk, which would be used to assess the risk of application(s) associated with the first loan that he/she applies. The second is a repeat or behaviour risk model, in which case the customer has been a client and applies for a repeat loan. In the latter case - we will have additional performance on how he/she repaid their prior loans, which we can incorporate into our risk model.

It is your job to predict if a loan was good or bad, i.e. accurately predict binary outcome variable, where Good is 1 and Bad is 0.

Data

We have three datasets: 1) traindemographics.csv : This holds customer's demographic data.

2) trainperf.csv : This holds data of the repeat loan that the customer has taken for which we need to predict the performance of.

3) trainprevloans.csv : This dataset contains all previous loans that the customer had prior to the loan above that we want to predict the performance of.

Further descriptions of the dataset and the data contained in them is given at the time of loading the dataset in cells ahead.

In [131…
```python
import pandas as pd
import warnings
import sys
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime, date
import numpy as np
from googlemaps import Client as GoogleMaps
import folium
from folium import Choropleth, Circle, Marker
from folium.plugins import MarkerCluster
from geopy.geocoders import Nominatim
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.linear_model import LogisticRegression
import category_encoders as ce
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay, classification_report
from imblearn.over_sampling import SMOTE
from sklearn.neighbors import KNeighborsClassifier
import calendar
from sklearn.preprocessing import binarize
from sklearn.svm import SVC
```

In [131…
```python
#preventing any warning messages from displaying during code execution
warnings.filterwarnings('ignore')
```

Next, the datasets are stored in variables in one cell and the next cell then holds the description of the data.

In [131…
```python
#loading traindemographics.csv into a variable called demographic
demographic = pd.read_csv('traindemographics.csv')
demographic.head()
```

Out[1315]:

	customerid	birthdate	bank_account_type	longitude_gps	latitude_gps	bank_name_clients	bank_branch_clients	employment_status_clients	level_of_education_clients
0	8a858e135cb22031015cbafc76964ebd	1973-10-10 00:00:00.000000	Savings	3.319219	6.528604	GT Bank	NaN	NaN	NaN
1	8a858e275c7ea5ec015c82482d7c3996	1986-01-21 00:00:00.000000	Savings	3.325598	7.119403	Sterling Bank	NaN	Permanent	NaN
2	8a858e5b5bd99460015bdc95cd485634	1987-04-01 00:00:00.000000	Savings	5.746100	5.563174	Fidelity Bank	NaN	NaN	NaN
3	8a858efd5ca70688015cabd1f1e94b55	1991-07-19 00:00:00.000000	Savings	3.362850	6.642485	GT Bank	NaN	Permanent	NaN
4	8a858e785acd3412015acd48f4920d04	1982-11-22 00:00:00.000000	Savings	8.455332	11.971410	GT Bank	NaN	Permanent	NaN

Description of demographic data:

- customerid (Primary key used to merge to other data)
- birthdate (date of birth of the customer)
- bank_account_type (type of primary bank account)
- longitude_gps
- latitude_gps
- bank_name_clients (name of the bank)
- bank_branch_clients (location of the branch - not compulsory - so missing in a lot of the cases)
- employment_status_clients (type of employment that customer has)
- level_of_education_clients (highest level of education)

In [131…

```
#loading trainperf.csv into a variable called performance
performance = pd.read_csv('trainperf.csv')
performance.head()
```

Out[1316]:

	customerid	systemloanid	loannumber	approveddate	creationdate	loanamount	totaldue	termdays	referredby	good_bad_flag
0	8a2a81a74ce8c05d014cfb32a0da1049	301994762	12	2017-07-25 08:22:56.000000	2017-07-25 07:22:47.000000	30000.0	34500.0	30	NaN	Good
1	8a85886e54beabf90154c0a29ae757c0	301965204	2	2017-07-05 17:04:41.000000	2017-07-05 16:04:18.000000	15000.0	17250.0	30	NaN	Good
2	8a8588f35438fe12015444567666018e	301966580	7	2017-07-06 14:52:57.000000	2017-07-06 13:52:51.000000	20000.0	22250.0	15	NaN	Good
3	8a85890754145ace015429211b513e16	301999343	3	2017-07-27 19:00:41.000000	2017-07-27 18:00:35.000000	10000.0	11500.0	15	NaN	Good
4	8a858970548359cc0154883481981866	301962360	9	2017-07-03 23:42:45.000000	2017-07-03 22:42:39.000000	40000.0	44000.0	30	NaN	Good

Performance data (trainperf.csv) : This is the repeat loan that the customer has taken for which we need to predict the performance of. Basically, we need to predict whether this loan would default given all previous loans and demographics of a customer. Description of data:

- customerid (Primary key used to merge to other data)
- systemloanid (The id associated with the particular loan. The same customerId can have multiple systemloanid's for each loan he/she has taken out)
- loannumber (The number of the loan that you have to predict)
- approveddate (Date that loan was approved)
- creationdate (Date that loan application was created)
- loanamount (Loan value taken)
- totaldue (Total repayment required to settle the loan - this is the capital loan value disbursed +interest and fees)
- termdays (Term of loan)
- referredby (customerId of the customer that referred this person - is missing, then not referred)
- good_bad_flag (good = settled loan on time; bad = did not settled loan on time) - this is the target variable that we need to predict

In [131…

```
#loading trainprevloans.csv into a variable called demographic
previous = pd.read_csv('trainprevloans.csv')
previous.head()
```

Out[1317]:

	customerid	systemloanid	loannumber	approveddate	creationdate	loanamount	totaldue	termdays	closeddate	referredby	firstduedate	firstrepaiddate
0	8a2a81a74ce8c05d014cfb32a0da1049	301682320	2	2016-08-15 18:22:40.000000	2016-08-15 17:22:32.000000	10000.0	13000.0	30	2016-09-01 16:06:48.000000	NaN	2016-09-14 00:00:00.000000	2016-09-01 15:51:43.000000
1	8a2a81a74ce8c05d014cfb32a0da1049	301883808	9	2017-04-28 18:39:07.000000	2017-04-28 17:38:53.000000	10000.0	13000.0	30	2017-05-28 14:44:49.000000	NaN	2017-05-30 00:00:00.000000	2017-05-26 00:00:00.000000
2	8a2a81a74ce8c05d014cfb32a0da1049	301831714	8	2017-03-05 10:56:25.000000	2017-03-05 09:56:19.000000	20000.0	23800.0	30	2017-04-26 22:18:56.000000	NaN	2017-04-04 00:00:00.000000	2017-04-26 22:03:47.000000
3	8a8588f35438fe12015444567666018e	301861541	5	2017-04-09 18:25:55.000000	2017-04-09 17:25:42.000000	10000.0	11500.0	15	2017-04-24 01:35:52.000000	NaN	2017-04-24 00:00:00.000000	2017-04-24 00:48:43.000000
4	8a85890754145ace015429211b513e16	301941754	2	2017-06-17 09:29:57.000000	2017-06-17 08:29:50.000000	10000.0	11500.0	15	2017-07-14 21:18:43.000000	NaN	2017-07-03 00:00:00.000000	2017-07-14 21:08:35.000000

Previous loans data (trainprevloans.csv) : This dataset contains all previous loans that the customer had prior to the loan above that we want to predict the performance of. Each loan will have a different systemloanid, but the same customerid for each customer. Description of data:

- customerid (Primary key used to merge to other data)
- systemloanid (The id associated with the particular loan. The same customerId can have multiple systemloanid's for each loan he/she has taken out)
- loannumber (The number of the loan that you have to predict)
- approveddate (Date that loan was approved)
- creationdate (Date that loan application was created)
- loanamount (Date that loan application was created)
- totaldue (Total repayment required to settle the loan - this is the capital loan value disbursed +interest and fees) termdays (Term of loan)
- closeddate (Date that the loan was settled)
- referredby (customerId of the customer that referred this person - is missing, then not refrerred)
- firstduedate (Date of first payment due in cases where the term is longer than 30 days. So in the case where the term is 60+ days - then there are multiple monthly payments due - and this dates reflects the date of the first payment)
- firstrepaiddate (Actual date that he/she paid the first payment as defined above)

We have 3 separate datasets and the way forward would be to join the three. But we need to know if it can be a straightforward action.

In the next cell, we check to see if all customers recorded in the demographic dataset exist in the other two datasets.

The result of the check is stored in 'customerCheck' variable.

In [131…

```
customerCheck = demographic.assign(InPerformance=demographic.customerid.isin(performance.customerid), InPrevious=demographic.customerid.isin(previous.customerid))
customerCheck = customerCheck[['customerid','InPerformance','InPrevious']]
```

In [131…

```
demographic.assign(InPerformance=demographic.customerid.isin(performance.customerid), InPrevious=demographic.customerid.isin(previous.customerid))
```

Out[1319]:

	customerid	birthdate	bank_account_type	longitude_gps	latitude_gps	bank_name_clients	bank_branch_clients	employment_status_clients	level_of_education_clients	InPerformance	InPrevious
0	8a858e135cb22031015cbafc76964ebd	1973-10-10 00:00:00.000000	Savings	3.319219	6.528604	GT Bank	NaN	NaN	NaN	True	True
1	8a858e275c7ea5ec015c82482d7c3996	1986-01-21 00:00:00.000000	Savings	3.325598	7.119403	Sterling Bank	NaN	Permanent	NaN	True	True
2	8a858e5b5bd99460015bdc95cd485634	1987-04-01 00:00:00.000000	Savings	5.746100	5.563174	Fidelity Bank	NaN	NaN	NaN	True	True
3	8a858efd5ca70688015cabd1f1e94b55	1991-07-19 00:00:00.000000	Savings	3.362850	6.642485	GT Bank	NaN	Permanent	NaN	True	True
4	8a858e785acd3412015acd48f4920d04	1982-11-22 00:00:00.000000	Savings	8.455332	11.971410	GT Bank	NaN	Permanent	NaN	False	False
...	...	...	...	...	...	...	...	...	...	...	...
4341	8a858f155554552501555588ca2b3b40	1985-12-13 00:00:00.000000	Other	3.236753	7.030168	Stanbic IBTC	NaN	Permanent	Graduate	True	True
4342	8a858fc65cf978f4015cf97cee3a02ce	1982-07-01 00:00:00.000000	Savings	7.013750	4.875662	GT Bank	NaN	NaN	NaN	True	True
4343	8a858f4f5b66de3a015b66fc83c61902	1989-09-26 00:00:00.000000	Savings	6.295530	7.092508	GT Bank	NaN	Permanent	NaN	False	False
4344	8aaae7a74400b28201441c8b62514150	1985-09-06 00:00:00.000000	Savings	3.354206	6.539070	GT Bank	HEAD OFFICE	Permanent	Primary	False	False
4345	8a85896653e2e18b0153e69c1b90265c	1975-06-05 00:00:00.000000	Savings	6.661014	7.472700	UBA	NaN	Permanent	NaN	False	False

4346 rows × 11 columns

In [132…

```
customerCheck
```

Out[1320]:

	customerid	InPerformance	InPrevious
0	8a858e135cb22031015cbafc76964ebd	True	True
1	8a858e275c7ea5ec015c82482d7c3996	True	True
2	8a858e5b5bd99460015bdc95cd485634	True	True
3	8a858efd5ca70688015cabd1f1e94b55	True	True
4	8a858e785acd3412015acd48f4920d04	False	False
...	...	...	...
4341	8a858f155554552501555588ca2b3b40	True	True
4342	8a858fc65cf978f4015cf97cee3a02ce	True	True
4343	8a858f4f5b66de3a015b66fc83c61902	False	False
4344	8aaae7a74400b28201441c8b62514150	False	False
4345	8a85896653e2e18b0153e69c1b90265c	False	False

4346 rows × 3 columns

From the display above of the contents of the 'customercheck' variable, we can see for each customer in the 'demographic' dataset, there are two boolean values associated with it.

The 'Inperformance' column is for the dataset performance and the 'InPrevious' column is for the dataset previous. 'True' implies that record is in a dataset corresponding to that column and 'False' implies that is not. For the records shown, so far we only see 'True,True' and 'False,False' pairs. Let us confirm that with a groupby applied to the 'customercheck' dataframe.

In [132…

```
customerCheck.groupby(['InPerformance','InPrevious']).value_counts()
```

Out[1321]:
```
InPerformance  InPrevious  customerid
False          False       8a858edd57f790040157ffe9b6ed3fbb    2
                           8a858f965bb63a25015bbf63fd062e2e    2
                           8a858fca5c35df2c015c39ad8695343e    2
                           8a858e625c8d993a015c938f829f77ee    2
                           8a28afc7474813a40147639ec637156b    1
                                                              ..
True           True        8a858e105bd92644015bd9db3a0f3be2    1
                           8a858e105bd92644015bdca43c877d2c    1
                           8a858e105bd92644015bdd2f7a981936    1
                           8a858e105bd92644015bdd374f0d1a3a    1
                           8a858fff5c79144c015c7bdbfc086ce1    1
Length: 4334, dtype: int64
```

From the groupby results above, we can see that in the 'customercheck' dataframe, there are only two existing pairs, 'True,True' and 'False,False'. This means there are only two possible scenarios for a customer:

1) A customer that has previously made loans(True,True) 2) A customer that has not taken out a loan before (False, False)

The records which have counts of 2 are most likely duplicates.

Next, we look at the more information about the dataframes and the data it contains

In [132...
```
demographic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4346 entries, 0 to 4345
Data columns (total 9 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   customerid                  4346 non-null   object
 1   birthdate                   4346 non-null   object
 2   bank_account_type           4346 non-null   object
 3   longitude_gps               4346 non-null   float64
 4   latitude_gps                4346 non-null   float64
 5   bank_name_clients           4346 non-null   object
 6   bank_branch_clients         51 non-null     object
 7   employment_status_clients   3698 non-null   object
 8   level_of_education_clients  587 non-null    object
dtypes: float64(2), object(7)
memory usage: 305.7+ KB
```

From the summary above of the 'demographic' dataframe, we can see that there are 4346 entries and with missing values in two of the columns:employment_status_clients and bank_branch_clients. There are only 2 numerical columns out of the 9 columns.

In [132...
```
performance.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4368 entries, 0 to 4367
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   customerid     4368 non-null   object
 1   systemloanid   4368 non-null   int64
 2   loannumber     4368 non-null   int64
 3   approveddate   4368 non-null   object
 4   creationdate   4368 non-null   object
 5   loanamount     4368 non-null   float64
 6   totaldue       4368 non-null   float64
 7   termdays       4368 non-null   int64
 8   referredby     587 non-null    object
 9   good_bad_flag  4368 non-null   object
dtypes: float64(2), int64(3), object(5)
memory usage: 341.4+ KB
```

From the summary above for the 'performance' dataframe, we can see that there are 4368 entries and only one column has missing values:referredby. There are 5 numerical columns and 5 categorical columns.

In [132...
```
previous.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18183 entries, 0 to 18182
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   customerid       18183 non-null  object 
 1   systemloanid     18183 non-null  int64  
 2   loannumber       18183 non-null  int64  
 3   approveddate     18183 non-null  object 
 4   creationdate     18183 non-null  object 
 5   loanamount       18183 non-null  float64
 6   totaldue         18183 non-null  float64
 7   termdays         18183 non-null  int64  
 8   closeddate       18183 non-null  object 
 9   referredby       1026 non-null   object 
 10  firstduedate     18183 non-null  object 
 11  firstrepaiddate  18183 non-null  object 
dtypes: float64(2), int64(3), object(7)
memory usage: 1.7+ MB
```

From the summary above for the 'previous' dataframe, we can see this dataframe has the most records: 18183. Only one column has missing values: referred by. There are five numerical columns out of the twelve columns.

From the summaries above, we saw that the 'previous' dataframe has the most entries out of all the dataframes. From the groupby done earlier on the customer check as well, we could see that if a customer exists in the previous dataframe, then they also exist in the performance dataframe. If there is anything the number of entries is telling us between these two dataframes is that there is one to many relationship between these two dataframes: previous dataframe is on the many side.

This can further be seen below when we obtain the records of customer whose id is '8a858e105bd92644015bd9db3a0f3be2' from both dataframes. The customer exists once in performance but twice in previous.

In [132…
```python
#records from performance dataframe of customer whose is '8a858e105bd92644015bd9db3a0f3be2'

performance.loc[performance['customerid'] == '8a858e105bd92644015bd9db3a0f3be2']
```

Out[1325]:

	customerid	systemloanid	loannumber	approveddate	creationdate	loanamount	totaldue	termdays	referredby	good_bad_flag
2899	8a858e105bd92644015bd9db3a0f3be2	301981450	3	2017-07-17 09:50:07.000000	2017-07-17 08:50:00.000000	10000.0	13000.0	30	NaN	Good

In [132…
```python
#records from previous dataframe of customer whose is '8a858e105bd92644015bd9db3a0f3be2'

previous.loc[previous['customerid'] == '8a858e105bd92644015bd9db3a0f3be2']
```

Out[1326]:

	customerid	systemloanid	loannumber	approveddate	creationdate	loanamount	totaldue	termdays	closeddate	referredby	firstduedate	firstrepaiddate
7765	8a858e105bd92644015bd9db3a0f3be2	301940743	2	2017-06-16 10:43:41.000000	2017-06-16 09:43:35.000000	10000.0	13000.0	30	2017-07-15 07:03:02.000000	NaN	2017-07-17 00:00:00.000000	2017-07-15 06:52:53.000000
12030	8a858e105bd92644015bd9db3a0f3be2	301911310	1	2017-05-17 13:00:00.000000	2017-05-17 11:58:51.000000	10000.0	13000.0	30	2017-06-16 10:36:38.000000	NaN	2017-06-16 00:00:00.000000	2017-06-16 10:26:29.000000

Checking for duplicates

We can check for duplicates in the dataframes

In [132…
```python
duplicate = demographic[demographic.duplicated()]

print("Duplicate Rows :")

# Print the resultant Dataframe
duplicate
```

```
Duplicate Rows :
```

Out[1327]:

	customerid	birthdate	bank_account_type	longitude_gps	latitude_gps	bank_name_clients	bank_branch_clients	employment_status_clients	level_of_education_clients
159	8a858fca5c35df2c015c39ad8695343e	1980-11-26 00:00:00.000000	Savings	3.352588	7.211089	GT Bank	NaN	Permanent	NaN
517	8a858edd57f790040157ffe9b6ed3fbb	1988-01-18 00:00:00.000000	Other	3.782563	7.171356	First Bank	NaN	Permanent	Secondary
776	8a858f965bb63a25015bbf63fd062e2e	1974-02-25 00:00:00.000000	Savings	3.936366	6.817958	Stanbic IBTC	NaN	Permanent	NaN
1015	8a858fe65675195a015679452588279c	1982-08-01 00:00:00.000000	Savings	7.533646	9.046885	UBA	NaN	Permanent	NaN
1090	8a858e6c5c88d145015c8b9627cd5a48	1979-09-30 00:00:00.000000	Savings	3.367008	6.497313	Sterling Bank	NaN	Permanent	NaN
1188	8a858fc75cd62882015cdaf2f4311b3f	1975-10-27 00:00:00.000000	Savings	7.437607	9.088935	GT Bank	NaN	Permanent	NaN
1480	8a858fe05d421ff4015d4c87d2a21ceb	1983-01-20 00:00:00.000000	Savings	8.526960	12.023015	Skye Bank	NaN	Permanent	NaN
1928	8a858e625c8d993a015c938f829f77ee	1988-12-20 00:00:00.000000	Savings	5.768333	5.561992	First Bank	NaN	Permanent	NaN
1996	8a858ec65cc6352b015cc64525ea0763	1985-01-30 00:00:00.000000	Savings	3.845728	7.411737	GT Bank	NaN	Permanent	NaN
4126	8a858f1e5baffcc9015bb02b505f180d	1983-04-06 00:00:00.000000	Savings	6.969350	4.818535	GT Bank	NaN	Permanent	NaN
4266	8a858f1e5cc4bc81015cc548e1eb5206	1979-09-15 00:00:00.000000	Savings	6.285242	4.922719	UBA	NaN	Permanent	NaN
4286	8a858f9f5679951a01567a5b90644817	1984-12-17 00:00:00.000000	Savings	4.196662	12.429509	Access Bank	NaN	Permanent	NaN

In [132…

```
duplicate = performance[performance.duplicated()]

print("Duplicate Rows :")

# Print the resultant Dataframe
duplicate
```

Duplicate Rows :

Out[1328]:

customerid	systemloanid	loannumber	approveddate	creationdate	loanamount	totaldue	termdays	referredby	good_bad_flag

In [132…

```
duplicate = previous[previous.duplicated()]

print("Duplicate Rows :")

# Print the resultant Dataframe
duplicate
```

Duplicate Rows :

Out[1329]:

customerid	systemloanid	loannumber	approveddate	creationdate	loanamount	totaldue	termdays	closeddate	referredby	firstduedate	firstrepaiddate

Only duplicates in the demographic dataframe and those will be dealt with below.

In [133…

```
demographic.drop_duplicates(inplace=True)
```

In [133…

```
customerCheck = demographic.assign(InPerformance=demographic.customerid.isin(performance.customerid), InPrevious=demographic.customerid.isin(previous.customerid))
customerCheck = customerCheck[['customerid','InPerformance','InPrevious']]
```

In [133…

```
#checking the value counts again and we can see that the earlier assumption of duplicates
#was true and now all records have only one count
customerCheck.groupby(['InPerformance','InPrevious']).value_counts()
```

Out[1332]:

```
InPerformance  InPrevious  customerid
False          False       8a28afc7474813a40147639ec637156b    1
                           8a3735d5518aba7301518ac34413010d    1
                           8a858f465668e3d60156790caa5f49da    1
                           8a858f3d5ab81f53015ab8334351169b    1
                           8a858f3d5add42e2015ae0d08df06d16    1
                                                              ..
True           True        8a858e0f5c45466f015c5469454d1227    1
                           8a858e0f5c45466f015c55c7e09d5e9c    1
                           8a858e105b434e9e015b437dbfab3920    1
                           8a858e105bd92644015bd9db3a0f3be2    1
                           8a858fff5c79144c015c7bdbfc086ce1    1
Length: 4334, dtype: int64
```

Depending on this project and its data, there is going to be a need for two models:

- the first model which would predict for non-new customers based on demographic and previous loans data.
- the second model which predict for new customers based on just demographic data.

As a group we are going to deal with and develop the first model.

Determining how the dataframes are going to be concatenated.

Checking the primary key columns between the data sets along which records will be connected

customerid

Performance dataframe has 18183 entries and previous has 4368 entries.

In [133…

```
performanceids = pd.DataFrame(performance['customerid'])
performanceids
```

Out[1333]:

	customerid
0	8a2a81a74ce8c05d014cfb32a0da1049
1	8a85886e54beabf90154c0a29ae757c0
2	8a8588f35438fe12015444567666018e
3	8a85890754145ace015429211b513e16
4	8a858970548359cc0154883481981866
...	...
4363	8a858e6d58b0cc520158beeb14b22a5a
4364	8a858ee85cf400f5015cf44ab1c42d5c
4365	8a858f365b2547f3015b284597147c94
4366	8a858f935ca09667015ca0ee3bc63f51
4367	8a858fd458639fcc015868eb14b542ad

4368 rows × 1 columns

In [133…

```
performanceids.groupby('customerid').value_counts().nlargest()
```

Out[1334]:

```
customerid
8a1088a0484472eb01484669e3ce4e0b    1
8a1a1e7e4f707f8b014f797718316cad    1
8a1a32fc49b632520149c3b8fdf85139    1
8a1eb5ba49a682300149c3c068b806c7    1
8a1edbf14734127f0147356fdb1b1eb2    1
dtype: int64
```

In [133…

```
previousids = pd.DataFrame(previous['customerid'])
previousids
```

Out[1335]:

	customerid
0	8a2a81a74ce8c05d014cfb32a0da1049
1	8a2a81a74ce8c05d014cfb32a0da1049
2	8a2a81a74ce8c05d014cfb32a0da1049
3	8a8588f35438fe12015444567666018e
4	8a85890754145ace015429211b513e16
...	...
18178	8a858899538ddb8e0153a2b555421fc5
18179	8a858899538ddb8e0153a2b555421fc5
18180	8a858899538ddb8e0153a2b555421fc5
18181	8a858f0656b7820c0156c92ca3ba436f
18182	8a858faf5679a838015688de3028143d

18183 rows × 1 columns

In [133…
```
previousids.groupby('customerid').value_counts().nlargest()
```

Out[1336]:
```
customerid
8a858f7d5578012a01557ea194d94948    26
8a858e4456ced8470156d73452f85335    22
8a85886f54517ee0015470749d3c3ce7    21
8a85888c548fb3d50154947fe59c32cf    21
8a858899538ddb8e0153a780c56e34bb    21
dtype: int64
```

From the analysis above, one can see that the most times a customer appears in the performance data is once, whereas the most times a customer appears in the previous data is 26 times. This confirms it is ideal to add the 'performance' dataframe to the previous dataframe.

adding the performance dataframe to the previous loans dataframe

Between the two dataframes,there are a number of similar columns. To add the two dataframes together, the column names of the incoming dataframe,'performance', will be changed to be able to distinguish the columns.

In [133…
```
performance.columns
```

Out[1337]:
```
Index(['customerid', 'systemloanid', 'loannumber', 'approveddate',
       'creationdate', 'loanamount', 'totaldue', 'termdays', 'referredby',
       'good_bad_flag'],
      dtype='object')
```

In [133…
```
previous.columns
```

Out[1338]:
```
Index(['customerid', 'systemloanid', 'loannumber', 'approveddate',
       'creationdate', 'loanamount', 'totaldue', 'termdays', 'closeddate',
       'referredby', 'firstduedate', 'firstrepaiddate'],
      dtype='object')
```

From the cells above, we can see the similar columns: customerid, systemloanid,loannumber,approveddate,creationdate,loanamount,totaldue,termdays and referred by. All of these columns,apart from customerid and systemloanid, will be renamed now. Customerid and systemloanid will not be renamed because they are going to be dropped ahead.

In [133…
```
performance.columns = ['customerid','perfsystemloanid','perf_loannumber','perf_approveddate','perf_creationdate','perf_loanamount','perf_totaldue','perf_termdays','perf_referredby','good_bad_fl
```

In [134…
```
performance.columns
```

Out[1340]:
```
Index(['customerid', 'perfsystemloanid', 'perf_loannumber',
       'perf_approveddate', 'perf_creationdate', 'perf_loanamount',
       'perf_totaldue', 'perf_termdays', 'perf_referredby', 'good_bad_flag'],
      dtype='object')
```

In [134…
```
previous.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18183 entries, 0 to 18182
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   customerid       18183 non-null  object 
 1   systemloanid     18183 non-null  int64  
 2   loannumber       18183 non-null  int64  
 3   approveddate     18183 non-null  object 
 4   creationdate     18183 non-null  object 
 5   loanamount       18183 non-null  float64
 6   totaldue         18183 non-null  float64
 7   termdays         18183 non-null  int64  
 8   closeddate       18183 non-null  object 
 9   referredby       1026 non-null   object 
 10  firstduedate     18183 non-null  object 
 11  firstrepaiddate  18183 non-null  object 
dtypes: float64(2), int64(3), object(7)
memory usage: 1.7+ MB
```

In [134…

```
#joining previous and performance dataframes in a dataframe called 'previousCustomers'
previousCustomers = pd.merge(previous, performance, on='customerid')
```

In [134…

```
previousCustomers
```

Out[1343]:

	customerid	systemloanid	loannumber	approveddate	creationdate	loanamount	totaldue	termdays	closeddate	referredby	...	firstrepaiddate	perfsystemloanid
0	8a2a81a74ce8c05d014cfb32a0da1049	301682320	2	2016-08-15 18:22:40.000000	2016-08-15 17:22:32.000000	10000.0	13000.0	30	2016-09-01 16:06:48.000000	NaN	...	2016-09-01 15:51:43.000000	301994762
1	8a2a81a74ce8c05d014cfb32a0da1049	301883808	9	2017-04-28 18:39:07.000000	2017-04-28 17:38:53.000000	10000.0	13000.0	30	2017-05-28 14:44:49.000000	NaN	...	2017-05-26 00:00:00.000000	301994762
2	8a2a81a74ce8c05d014cfb32a0da1049	301831714	8	2017-03-05 10:56:25.000000	2017-03-05 09:56:19.000000	20000.0	23800.0	30	2017-04-26 22:18:56.000000	NaN	...	2017-04-26 22:03:47.000000	301994762
3	8a2a81a74ce8c05d014cfb32a0da1049	301923941	10	2017-06-01 13:34:30.000000	2017-06-01 12:34:21.000000	20000.0	24500.0	30	2017-06-25 15:24:06.000000	NaN	...	2017-06-25 15:13:56.000000	301994762
4	8a2a81a74ce8c05d014cfb32a0da1049	301954468	11	2017-06-28 10:58:34.000000	2017-06-28 09:58:25.000000	20000.0	24500.0	30	2017-07-25 08:14:36.000000	NaN	...	2017-07-25 08:04:27.000000	301994762
...	...	...	...	...	...	...	...	...	...	...	...	...	...
18178	8a858f305c8dd672015c92b0711a3333	301941335	1	2017-06-16 18:16:37.000000	2017-06-16 17:15:29.000000	10000.0	11500.0	15	2017-06-26 14:02:03.000000	NaN	...	2017-06-26 13:51:54.000000	301978946
18179	8a858fe7568ed7420156920bff565cc7	301955570	1	2017-06-29 01:25:57.000000	2017-06-29 00:25:48.000000	10000.0	11500.0	15	2017-07-05 14:31:17.000000	NaN	...	2017-07-05 14:21:08.000000	301976025
18180	8a858f6459b6456d0159b69978f22bed	301796830	1	2017-01-19 14:00:16.000000	2017-01-19 13:00:02.000000	10000.0	11500.0	15	2017-02-15 09:06:34.000000	NaN	...	2017-02-15 08:51:25.000000	301969032
18181	8a858fad5ccb633e015ccbe337372ab3	301946936	1	2017-06-21 20:19:29.000000	2017-06-21 19:18:21.000000	10000.0	13000.0	30	2017-07-07 17:08:47.000000	8a858eaa55a0b8ae0155ad2cab5e49cc	...	2017-07-07 16:58:38.000000	301977456
18182	8a858f0656b7820c0156c92ca3ba436f	301697691	1	2016-08-27 20:03:45.000000	2016-08-27 19:03:34.000000	10000.0	13000.0	30	2016-10-15 10:17:54.000000	NaN	...	2016-10-15 10:02:45.000000	301996908

18183 rows × 21 columns

In [134…

```
previousCustomers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18183 entries, 0 to 18182
Data columns (total 21 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   customerid         18183 non-null  object 
 1   systemloanid       18183 non-null  int64  
 2   loannumber         18183 non-null  int64  
 3   approveddate       18183 non-null  object 
 4   creationdate       18183 non-null  object 
 5   loanamount         18183 non-null  float64
 6   totaldue           18183 non-null  float64
 7   termdays           18183 non-null  int64  
 8   closeddate         18183 non-null  object 
 9   referredby         1026 non-null   object 
 10  firstduedate       18183 non-null  object 
 11  firstrepaiddate    18183 non-null  object 
 12  perfsystemloanid   18183 non-null  int64  
 13  perf_loannumber    18183 non-null  int64  
 14  perf_approveddate  18183 non-null  object 
 15  perf_creationdate  18183 non-null  object 
 16  perf_loanamount    18183 non-null  float64
 17  perf_totaldue      18183 non-null  float64
 18  perf_termdays      18183 non-null  int64  
 19  perf_referredby    1026 non-null   object 
 20  good_bad_flag      18183 non-null  object 
dtypes: float64(4), int64(6), object(11)
memory usage: 3.1+ MB
```

Dropping the systemloanid columns as these are just transaction identifiers

In [134…

```
previousCustomers.drop(['systemloanid','perfsystemloanid'],axis=1,inplace=True)
previousCustomers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18183 entries, 0 to 18182
Data columns (total 19 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   customerid         18183 non-null  object 
 1   loannumber         18183 non-null  int64  
 2   approveddate       18183 non-null  object 
 3   creationdate       18183 non-null  object 
 4   loanamount         18183 non-null  float64
 5   totaldue           18183 non-null  float64
 6   termdays           18183 non-null  int64  
 7   closeddate         18183 non-null  object 
 8   referredby         1026 non-null   object 
 9   firstduedate       18183 non-null  object 
 10  firstrepaiddate    18183 non-null  object 
 11  perf_loannumber    18183 non-null  int64  
 12  perf_approveddate  18183 non-null  object 
 13  perf_creationdate  18183 non-null  object 
 14  perf_loanamount    18183 non-null  float64
 15  perf_totaldue      18183 non-null  float64
 16  perf_termdays      18183 non-null  int64  
 17  perf_referredby    1026 non-null   object 
 18  good_bad_flag      18183 non-null  object 
dtypes: float64(4), int64(4), object(11)
memory usage: 2.8+ MB
```

In [134…

```
#adding the demographic data to the previousCustomers dataframe
previousCustomers = pd.merge(previousCustomers, demographic, on='customerid')
```

In [134…

```
previousCustomers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13673 entries, 0 to 13672
Data columns (total 27 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   customerid                 13673 non-null  object 
 1   loannumber                 13673 non-null  int64  
 2   approveddate               13673 non-null  object 
 3   creationdate               13673 non-null  object 
 4   loanamount                 13673 non-null  float64
 5   totaldue                   13673 non-null  float64
 6   termdays                   13673 non-null  int64  
 7   closeddate                 13673 non-null  object 
 8   referredby                 800 non-null    object 
 9   firstduedate               13673 non-null  object 
 10  firstrepaiddate            13673 non-null  object 
 11  perf_loannumber            13673 non-null  int64  
 12  perf_approveddate          13673 non-null  object 
 13  perf_creationdate          13673 non-null  object 
 14  perf_loanamount            13673 non-null  float64
 15  perf_totaldue              13673 non-null  float64
 16  perf_termdays              13673 non-null  int64  
 17  perf_referredby            800 non-null    object 
 18  good_bad_flag              13673 non-null  object 
 19  birthdate                  13673 non-null  object 
 20  bank_account_type          13673 non-null  object 
 21  longitude_gps              13673 non-null  float64
 22  latitude_gps               13673 non-null  float64
 23  bank_name_clients          13673 non-null  object 
 24  bank_branch_clients        104 non-null    object 
 25  employment_status_clients  12310 non-null  object 
 26  level_of_education_clients 3464 non-null   object 
dtypes: float64(6), int64(4), object(17)
memory usage: 2.9+ MB
```

looking at the data contained in the columns now to determine what columns will be used to train the model

columns containing non-numerical data

In [134…

```python
categorical = [var for var in previousCustomers.columns if previousCustomers[var].dtype=='O']

print('There are {} categorical variables \n'.format(len(categorical)))

print('They are: ', categorical)
```

```
There are 17 categorical variables 

They are:  ['customerid', 'approveddate', 'creationdate', 'closeddate', 'referredby', 'firstduedate', 'firstrepaiddate', 'perf_approveddate', 'perf_creationdate', 'perf_referredby', 'good_bad_f
lag', 'birthdate', 'bank_account_type', 'bank_name_clients', 'bank_branch_clients', 'employment_status_clients', 'level_of_education_clients']
```

In [134…

```python
previousCustomers[categorical].head()
```

Out[1349]:

	customerid	approveddate	creationdate	closeddate	referredby	firstduedate	firstrepaiddate	perf_approveddate	perf_creationdate	perf_referredby	good_bad_flag	birthdate	bank_acc
0	8a2a81a74ce8c05d014cfb32a0da1049	2016-08-15 18:22:40.000000	2016-08-15 17:22:32.000000	2016-09-01 16:06:48.000000	NaN	2016-09-14 00:00:00.000000	2016-09-01 15:51:43.000000	2017-07-25 08:22:56.000000	2017-07-25 07:22:47.000000	NaN	Good	1972-01-15 00:00:00.000000	
1	8a2a81a74ce8c05d014cfb32a0da1049	2017-04-28 18:39:07.000000	2017-04-28 17:38:53.000000	2017-05-28 14:44:49.000000	NaN	2017-05-30 00:00:00.000000	2017-05-26 00:00:00.000000	2017-07-25 08:22:56.000000	2017-07-25 07:22:47.000000	NaN	Good	1972-01-15 00:00:00.000000	
2	8a2a81a74ce8c05d014cfb32a0da1049	2017-03-05 10:56:25.000000	2017-03-05 09:56:19.000000	2017-04-26 22:18:56.000000	NaN	2017-04-04 00:00:00.000000	2017-04-26 22:03:47.000000	2017-07-25 08:22:56.000000	2017-07-25 07:22:47.000000	NaN	Good	1972-01-15 00:00:00.000000	
3	8a2a81a74ce8c05d014cfb32a0da1049	2017-06-01 13:34:30.000000	2017-06-01 12:34:21.000000	2017-06-25 15:24:06.000000	NaN	2017-07-03 00:00:00.000000	2017-06-25 15:13:56.000000	2017-07-25 08:22:56.000000	2017-07-25 07:22:47.000000	NaN	Good	1972-01-15 00:00:00.000000	
4	8a2a81a74ce8c05d014cfb32a0da1049	2017-06-28 10:58:34.000000	2017-06-28 09:58:25.000000	2017-07-25 08:14:36.000000	NaN	2017-07-31 00:00:00.000000	2017-07-25 08:04:27.000000	2017-07-25 08:22:56.000000	2017-07-25 07:22:47.000000	NaN	Good	1972-01-15 00:00:00.000000	

customer id column: Primary key used to merge to other data

In [135…
```
len(previousCustomers['customerid'].unique())
```
Out[1350]:
```
3264
```

customer id column has a high cardinality and is used to uniquely identify the customers. This feature will not be used for model training.

approved date column : Date that loan was approved

In [135…
```
len(previousCustomers['approveddate'].unique())
```
Out[1351]:
```
13666
```

approved date column has a high cardinality. It will not be used because we don't believe the approval data for a loan holds a significance towards whether a loan may be bad or good.

creation date column : Date that loan application was created

In [135…
```
len(previousCustomers['creationdate'].unique())
```
Out[1352]:
```
13666
```

creation date column has a high cardinality. It will not be selected as feature to train the model because we do not believe the creation date for a loan holds a significance towards whether a loan may be bad or good

closed date column : Date that the loan was settled

closed date column has a high cardinality. Since we have a loan creation date, we can arrive at how long the loan was active until it was settled. This time period can then be used as a new feature to train the model. Let's look more at the closed dates and see if there is a relationship between the closing dates and good/bad loans.

In [135…
```
#extracting the closed date of customer's loans
closedDates = previousCustomers[['closeddate','good_bad_flag']]
closedDates
```
Out[1353]:

	closeddate	good_bad_flag
0	2016-09-01 16:06:48.000000	Good
1	2017-05-28 14:44:49.000000	Good
2	2017-04-26 22:18:56.000000	Good
3	2017-06-25 15:24:06.000000	Good
4	2017-07-25 08:14:36.000000	Good
...	...	...
13668	2017-07-18 16:33:55.000000	Good
13669	2017-07-11 14:26:40.000000	Good
13670	2017-06-26 14:02:03.000000	Good
13671	2017-07-05 14:31:17.000000	Good
13672	2017-02-15 09:06:34.000000	Good

13673 rows × 2 columns

going to break down the date and individual components, ie year, month and day and time. Then proceed to gain insights into the relationship between date/time and bad/good loans

In [135…
```
#converting 'closeddate' columns into datetime object
closedDates['closeddate'] = pd.to_datetime(closedDates['closeddate'])
```

In [135…
```
#checking to see the closeddate column datatype
closedDates.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13673 entries, 0 to 13672
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   closeddate     13673 non-null  datetime64[ns]
 1   good_bad_flag  13673 non-null  object
dtypes: datetime64[ns](1), object(1)
memory usage: 320.5+ KB
```

In [135…

```
#extracting the year from closeddate column
closedDates['year'] = closedDates['closeddate'].dt.year
closedDates.head()
```

Out[1356]:

	closeddate	good_bad_flag	year
0	2016-09-01 16:06:48	Good	2016
1	2017-05-28 14:44:49	Good	2017
2	2017-04-26 22:18:56	Good	2017
3	2017-06-25 15:24:06	Good	2017
4	2017-07-25 08:14:36	Good	2017

In [135…

```
#extracting the month from closeddate column
closedDates['month'] = closedDates['closeddate'].dt.month
closedDates.head()
```

Out[1357]:

	closeddate	good_bad_flag	year	month
0	2016-09-01 16:06:48	Good	2016	9
1	2017-05-28 14:44:49	Good	2017	5
2	2017-04-26 22:18:56	Good	2017	4
3	2017-06-25 15:24:06	Good	2017	6
4	2017-07-25 08:14:36	Good	2017	7

In [135…

```
#extracting the day from closeddate column
closedDates['day'] = closedDates['closeddate'].dt.day
closedDates.head()
```

Out[1358]:

	closeddate	good_bad_flag	year	month	day
0	2016-09-01 16:06:48	Good	2016	9	1
1	2017-05-28 14:44:49	Good	2017	5	28
2	2017-04-26 22:18:56	Good	2017	4	26
3	2017-06-25 15:24:06	Good	2017	6	25
4	2017-07-25 08:14:36	Good	2017	7	25

In [135…

```
#extracting the hour from closeddate column
closedDates['hourOfDay'] = closedDates['closeddate'].dt.hour
closedDates.head()
```

Out[1359]:

	closeddate	good_bad_flag	year	month	day	hourOfDay
0	2016-09-01 16:06:48	Good	2016	9	1	16
1	2017-05-28 14:44:49	Good	2017	5	28	14
2	2017-04-26 22:18:56	Good	2017	4	26	22
3	2017-06-25 15:24:06	Good	2017	6	25	15
4	2017-07-25 08:14:36	Good	2017	7	25	8

In [136…

```
#checking to see if the hours are in 12-hour or 24-hour format
closedDates['hourOfDay'].unique()
```

Out[1360]:

```
array([16, 14, 22, 15,  8, 13, 18, 11,  1,  4,  0, 10, 17, 21, 23,  9, 20,
       12,  5,  6, 19,  7,  3,  2])
```

the hour is recorded in 24 hour clock which is desirable so we can differentiate between the times i.e 3 in the morning and 3 in the evening.

In [136…

```
closedDates['year'].unique()
```

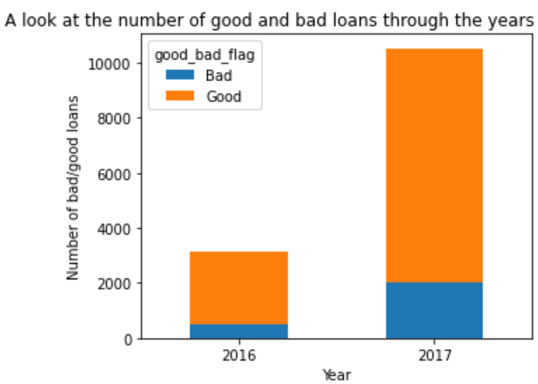
Out[1361]:

```
array([2016, 2017])
```

This dataset contains records for only the years 2016 and 2017.

In [136…

```
#plotting a stacked bar graph to show the loans(good and bad) in the years 2016 and 2017

closedDates.groupby(['year','good_bad_flag']).size().unstack().plot(kind='bar',stacked=True,rot=0)
plt.xlabel('Year')
plt.ylabel('Number of bad/good loans')

plt.title('A look at the number of good and bad loans through the years 2016 and 2017')

#getting current figure instance and setting width
f = plt.gcf()
f.set_figwidth(5)
```

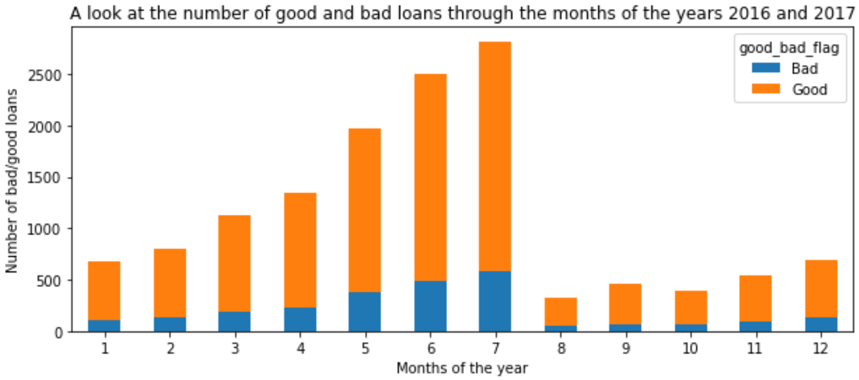


From the graph above, we can see that the number of bad loans increased from 2016 to 2017 as well as the overall number of loans being taken out

From this graph as well we can notice that our data is imbalance with the good loans outnumbering the bad loans by a large margin. This will be dealt on ahead in the notebook.

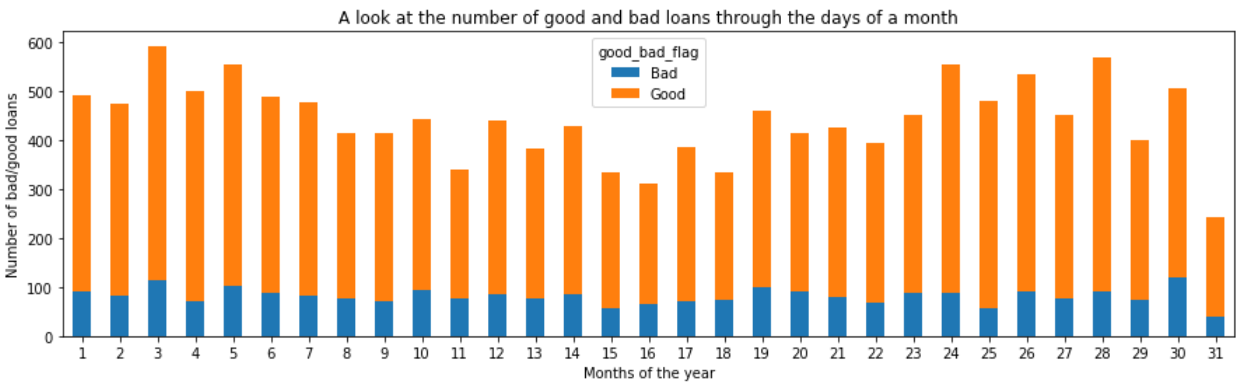
In [136…

```
closedDates.groupby(['month','good_bad_flag']).size().unstack().plot(kind='bar',stacked=True,rot=0)
plt.xlabel('Months of the year')
plt.ylabel('Number of bad/good loans')
plt.title('A look at the number of good and bad loans through the months of the years 2016 and 2017')

f = plt.gcf()
f.set_figwidth(10)
```



From the graph above, we can see that for both the years 2016 and 2017, there was a high number of loans, good and bad in July and then the least number in the subsequent month of August

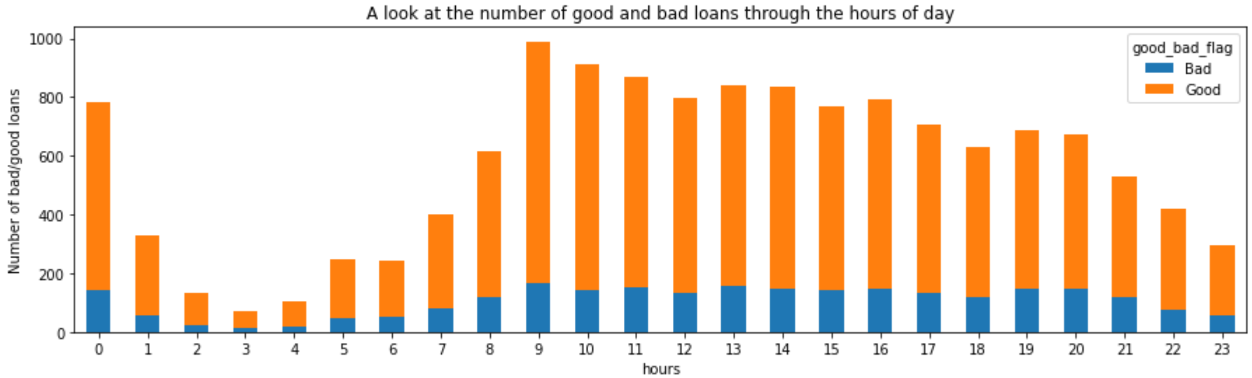
In [136…
```
closedDates.groupby(['day','good_bad_flag']).size().unstack().plot(kind='bar',stacked=True,rot=0)
plt.xlabel('Months of the year')
plt.ylabel('Number of bad/good loans')
plt.title('A look at the number of good and bad loans through the days of a month')

f = plt.gcf()
f.set_figwidth(15)
```



From the graph above, we can see there is not much to the type of loans according to the period in the month: beginning, end or middle

In [136…
```
plt.figure(figsize=(15,30))
closedDates.groupby(['hourOfDay','good_bad_flag']).size().unstack().plot(kind='bar',stacked=True,rot=0)
plt.xlabel('hours')
plt.ylabel('Number of bad/good loans')

plt.title('A look at the number of good and bad loans through the hours of day')

f = plt.gcf()
f.set_figwidth(15)
```

```
<Figure size 1080x2160 with 0 Axes>
```



From the graph above, we can see that there is not much activity between the hours of 1 and 6 since people are sleeping and for the rest of the day, nothing much significant in the loan activity.

We do not think that the year serves as a significant feature to train this model and therefore it will not be used going forward. The months, day and hour of the day also do not serve as good features to use to train this model.

In [136…

```
previousCustomers.head()
```

Out[1366]:

	customerid	loannumber	approveddate	creationdate	loanamount	totaldue	termdays	closeddate	referredby	firstduedate	...	perf_referredby	good_bad_flag	birthdate	bank_accoun
0	8a2a81a74ce8c05d014cfb32a0da1049	2	2016-08-15 18:22:40.000000	2016-08-15 17:22:32.000000	10000.0	13000.0	30	2016-09-01 16:06:48.000000	NaN	2016-09-14 00:00:00.000000	...	NaN	Good	1972-01-15 00:00:00.000000	
1	8a2a81a74ce8c05d014cfb32a0da1049	9	2017-04-28 18:39:07.000000	2017-04-28 17:38:53.000000	10000.0	13000.0	30	2017-05-28 14:44:49.000000	NaN	2017-05-30 00:00:00.000000	...	NaN	Good	1972-01-15 00:00:00.000000	
2	8a2a81a74ce8c05d014cfb32a0da1049	8	2017-03-05 10:56:25.000000	2017-03-05 09:56:19.000000	20000.0	23800.0	30	2017-04-26 22:18:56.000000	NaN	2017-04-04 00:00:00.000000	...	NaN	Good	1972-01-15 00:00:00.000000	
3	8a2a81a74ce8c05d014cfb32a0da1049	10	2017-06-01 13:34:30.000000	2017-06-01 12:34:21.000000	20000.0	24500.0	30	2017-06-25 15:24:06.000000	NaN	2017-07-03 00:00:00.000000	...	NaN	Good	1972-01-15 00:00:00.000000	
4	8a2a81a74ce8c05d014cfb32a0da1049	11	2017-06-28 10:58:34.000000	2017-06-28 09:58:25.000000	20000.0	24500.0	30	2017-07-25 08:14:36.000000	NaN	2017-07-31 00:00:00.000000	...	NaN	Good	1972-01-15 00:00:00.000000	

5 rows × 27 columns

We can instead obtain the duration of the lifetime of the loan i.e from when it was created to when it was fully paid back.

In [136…

```
previousCustomers['closeddate'] = pd.to_datetime(previousCustomers['closeddate'])
previousCustomers['creationdate'] = pd.to_datetime(previousCustomers['creationdate'])
```

In [136…

```
#the difference between the closedDate and the creationDate is stored in a new column
# called 'loanlife'. The difference between the two columns is divided by a delta object
# so we can get the difference in days
previousCustomers['loanlife'] = ((previousCustomers.closeddate - previousCustomers.creationdate)/np.timedelta64(1, 'D'))
previousCustomers.head()
```

Out[1368]:

	customerid	loannumber	approveddate	creationdate	loanamount	totaldue	termdays	closeddate	referredby	firstduedate	...	good_bad_flag	birthdate	bank_account_type	longitude_gps
0	8a2a81a74ce8c05d014cfb32a0da1049	2	2016-08-15 18:22:40.000000	2016-08-15 17:22:32	10000.0	13000.0	30	2016-09-01 16:06:48	NaN	2016-09-14 00:00:00.000000	...	Good	1972-01-15 00:00:00.000000	Other	3.43201
1	8a2a81a74ce8c05d014cfb32a0da1049	9	2017-04-28 18:39:07.000000	2017-04-28 17:38:53	10000.0	13000.0	30	2017-05-28 14:44:49	NaN	2017-05-30 00:00:00.000000	...	Good	1972-01-15 00:00:00.000000	Other	3.43201
2	8a2a81a74ce8c05d014cfb32a0da1049	8	2017-03-05 10:56:25.000000	2017-03-05 09:56:19	20000.0	23800.0	30	2017-04-26 22:18:56	NaN	2017-04-04 00:00:00.000000	...	Good	1972-01-15 00:00:00.000000	Other	3.43201
3	8a2a81a74ce8c05d014cfb32a0da1049	10	2017-06-01 13:34:30.000000	2017-06-01 12:34:21	20000.0	24500.0	30	2017-06-25 15:24:06	NaN	2017-07-03 00:00:00.000000	...	Good	1972-01-15 00:00:00.000000	Other	3.43201
4	8a2a81a74ce8c05d014cfb32a0da1049	11	2017-06-28 10:58:34.000000	2017-06-28 09:58:25	20000.0	24500.0	30	2017-07-25 08:14:36	NaN	2017-07-31 00:00:00.000000	...	Good	1972-01-15 00:00:00.000000	Other	3.43201

5 rows × 28 columns

We will use the loanlife column as a new feature to train our model with.

We are going to list out the columns that have been selected to train our model to keep track of them as the notebook progresses:

1. period take to settle the loan(lifetime of loan)

referredby column

In [136…
```
previousCustomers['referredby'].isnull().sum()
```

Out[1369]:
```
12873
```

In [137…
```
len(previousCustomers['referredby'].unique())
```

Out[1370]:
```
408
```

In [137…
```
previousCustomers['referredby'].unique()
```

```
Out[1371]: array([nan, '8a858ff85bd93919015bda56605652bd',
       '8a858edc59ee87640159eeb9774f1aa7',
       '8a858fc55b2548dd015b286e452c678c',
       '8a858eba5c884d2a015c8bea59385157',
       '8a858e185b4923b4015b536354895cae',
       '8a858fd85b685607015b7ba5638674c0',
       '8a858f9f5bd99987015be3f53cb442d9',
       '8a858fee5800e000015805aae2fe4ef8',
       '8a858fdc57ab280a0157b277e9ff6d53',
       '8a858e645755e62201575e72c8b77fd5',
       '8a858edb5ad79cc6015ad7c9b29415c8',
       '8a858f975c4582c4015c54619e9b7e2b',
       '8a858e6f5bce1023015bce19dbbf038d',
       '8a858f1b5b9136f7015b9f66e5c5327c',
       '8a858e7855113fb701551fe018ce6913',
       '8a858fa3551e78cb01552596fed60cec',
       '8a858eda5c8863ff015c8b96e5fd74d0',
       '8a858ed75732b681015744c9dee21aae',
       '8a858f1e5aec5791015aef876abb0dbc',
       '8a858fa75b5c94ed015b5c9d50890bb4',
       '8a8589ac53a917ed0153a9f9e614244b',
       '8a858f475c7ea023015c81ad999a0386',
       '8a858e5b5bd99460015bd9e9ee2e18f4',
       '8a8588dd54be35520154c9ffe1f03738',
       '8a858e6a5bfd4b54015c0297d9ee75fb',
       '8a858f185c884965015c88980e642dc1',
       '8a858edb5bd936e4015be394167c4698',
       '8a858f495bcefdb9015bcf9d6b173bf5',
       '8a858f6e5b1a4fd0015b1a555f54042b',
       '8a858f3b5c1c5a08015c1f69e3786a85',
       '8a858fa05908043801590d3c3a4e6f83',
       '8a858f7d5cf9aef9015cff00020175f0',
       '8a8589f35451855401547fc55caa6cd5',
       '8a858f1e5add5268015aea11ac2871cf',
       '8a858f8e5b8c34d3015b8cb7246e3f40',
       '8a858f155b24e303015b24f3a07610b5',
       '8a858f275c451af5015c52e90bb11bfe',
       '8a858e105bd92644015bdaea61965690',
       '8a858e4b5a94ae68015a9a8f07ab3b6a',
       '8a858f275c451af5015c54543f824e92',
       '8a858f5d5963fbb701596519a1b74190',
       '8a858fd35b5dfefb015b5f002aa050a3',
       '8a858ec55b441639015b46eaa7c63973',
       '8a858fed5a28bf0e015a29acfb4b3fcf',
       '8a858e105bd92644015bdc54519566e5',
       '8a858f855bf2f1ad015bf5c0e6c01b8a',
       '8a858fed5a3d7bb7015a3de90f6c297c',
       '8a858ee05bcdea36015bce28178318af',
       '8a858f725b49c0c0015b56d2ee6824d9',
       '8a858ff35acdd593015ad12db5593535',
       '8a858ede5ca210f0015ca32170645e53',
       '8a858f4e5ca72981015cab698acd7f40',
       '8a858fb85b81c658015b81c9ff880479',
       '8a858eac5c451942015c5635a22f58f6',
       '8a858e945bfdb9e7015c040d93016282',
       '8a858e2d5bb55d46015bc86b9e566836',
       '8a858f4c5bc9eb48015bca216c74199a',
       '8a858ef458d4a52e0158d858eb5d6456',
       '8a858fbf5b3a0b46015b3a75f2633e33',
       '8a858edb5bd936e4015be08d9d5a7916',
       '8a858e5f5a41b97f015a4212766e2f84',
       '8a858f895b1f9aec015b1ff32a664db3',
       '8a858e225c404292015c547de47a4ada',
       '8a858edc5c7e7da5015c81ea45c62462',
       '8a858edb552adcfd015531bdc5371d37',
       '8a858f535908095c01591ce9992f00c5',
       '8a858f7e5c69a6e3015c6d308889653b',
       '8a858eba5c884d2a015c8d20c864739e',
       '8a858fde56eb02280156eb911f1e3872',
       '8a858f045bb63525015bc84a44753992',
       '8a858f335c8d9723015c9c1a07d8681a',
```

'8a858e3a5bed83ab015bedad8d191c74',
'8a8588dc536a141301538365df9251b8',
'8a858f615578136301557e762f974b9f',
'8a858e885b68546c015b690a3b432fb4',
'8a858fb85b81c658015b8508fbb237a1',
'8a858e495d1e235c015d30d5ddb42149',
'8a858f615b1ffb14015b2316f57f3839',
'8a858f045bab240c015bae905a8e7527',
'8a858f725b49c0c0015b4d09b3876961',
'8a858f065b1fe50a015b236489d57fb6',
'8a858f1e5add5268015add535fe800a4',
'8a858f3d5b4411b7015b47d39a1f0401',
'8a858e135c7e2eec015c813211a96674',
'8a8589f853f100fb015402f057354ac0',
'8a858efe5bca0a81015bcdaee15c0bb3',
'8a858f585bfd6341015bfed6f96d6046',
'8a858e705b3a03c7015b3ceed0d531fb',
'8a858e4b5c831f02015c83bd0d353711',
'8a858f2c585375f00158678b108144f8',
'8a858fa4552add9f0155467c89c31293',
'8a858ffd5c8d343d015c8eae471c7b48',
'8a8589f853f100fb01540a3551f034bd',
'8a858f2b5b870cfe015b88b66b6673de',
'8a858e3c5cac9fcc015cafc54f557506',
'8a858f305c8dd672015c997141130c2e',
'8a858f965bb63a25015bc2ddf2577416',
'8a858f615591c2cc01559597a77202c6',
'8a858ff45be92ea2015be9db14023e1e',
'8a858f225b3dc49e015b3e084e3f5362',
'8a858f665c2142a0015c2cf209e20a6f',
'8a858e6d5b252065015b305a3ef77693',
'8a858e495cef1ef1015cef8066103757',
'8a858fe65675195a0156766ebdb042c8',
'8a858f8f5bfd3cfa015c026e474b47a5',
'8a858f665cf43442015cf4504cd91679',
'8a858f00590ac4e70159481263652a69',
'8a858fc55b2548dd015b28bb609670b2',
'8a858e395cb1d4d9015cc0480a8943bd',
'8a858f115b80b9dd015b81383fba4090',
'8a858f065cf41809015cf44c98422994',
'8a858ff9580b8812015815c32c7b1cd8',
'8a858fd75bb62bbe015bc52a13c36f6c',
'8a858ec358dc5c130158ea2c46c124be',
'8a858f9d5668e3f10156767e9a0f0188',
'8a858fa359d61dae0159d698bf503d29',
'8a858f685acd5951015acd609d0803c4',
'8a858ee758a4dfbc0158abd355eb4e87',
'8a858fc45c63d75d015c64d986e30084',
'8a858fbf5b3a0b46015b3ac9ef325d05',
'8a858e125b391784015b395fa56243e6',
'8a858ee05989e8dc01598a1083fa036e',
'8a858f615b910db3015b966a2a3d33f4',
'8a858fe45cd62cbf015ce67f3f26051a',
'8a858e215ca77c7f015ca871c384393d',
'8a858f2a5b3f2121015b415a99f4168b',
'8a8589ec542eaa8901544d30b510384d',
'8a858e6a5bfd4b54015c031f59e70807',
'8a858e375b347293015b347501e10206',
'8a858f225bfdb280015bfe6280983415',
'8a858f7d5578012a015579e9cfd56f51',
'8a858f755af1caee015af206036c172c',
'8a858899538ddb8e01539eccc3c375fe',
'8a858fe25b8c4e5c015b8f61faf34239',
'8a858f0056b7cf8e0156c14b37d37121',
'8a85884e54a0565b0154a05a7dd10159',
'8a858f995623749101562b51bcad0eca',
'8a858f1b5b256bc9015b2f998c4d6822',
'8a858fd45bf83e54015bf83efa9d004f',
'8a85880353acff880153af354a6719df',
'8a85889953a91b8d0153ac729ae74066',
'8a858e6c5bd3fbff015bd45bd46f2fbc',

'8a858f305cb1ec4e015cc41c66eb41b8',
'8a858f2c585375f00158688cf76b3db7',
'8a858eda552adcc2015548f0eb523fc3',
'8a858ee65be8d42d015be8ed0ebb0f5f',
'8a858e935b496584015b4a2c3e973ad3',
'8a858f365b2547f3015b2bfd85c85ec1',
'8a858e135cb22031015cb7f9700e06f8',
'8a858e725cf420fd015cf45addee2730',
'8a858f295c8d307f015c9b4d5c570061',
'8a858e275b349e25015b384ed5222fcb',
'8a858e025ab3a6c2015ab74cfa5c4431',
'8a85883353e2e4e40153eca57a9855c6',
'8a858fba5c1b5c10015c1b6bce4a075a',
'8a858f8658c206a90158c93edbb13b71',
'8a858e6d5b62e36f015b667380e92585',
'8a858fd75bb62bbe015bba3ca3b06158',
'8a858f4e5ba5e8d3015ba99033a10cd3',
'8a858ec25a4d1d16015a5004af0d4d20',
'8a858f3355ae6a610155c583cc8e034d',
'8a858f475c7ea023015c7ef5a10311c0',
'8a858fca5c35df2c015c39f8efd26149',
'8a858fa25bd92c20015bdbe4fac67f20',
'8a858f465668e3d601568add9c6a4fe0',
'8a858ebd5b5d50d9015b5d8c6c953f04',
'8a858e1a59cbf7460159d0335d9a6a03',
'8a858f7d5ab3ca4b015ab769621b5811',
'8a858f5b5bee1b11015bee3c86711191',
'8a858f165b147438015b149b3acc1b44',
'8a858f385b34d285015b368d23720716',
'8a858e195af64338015af6bc45ef5aef',
'8a858f1b5b256bc9015b2bd3050a1fc7',
'8a858f6e5a951fc2015a99d60afa0a68',
'8a858f375b493337015b4a7a04286bac',
'8a858f93572e873c01573e5728586df8',
'8a858e885c9c6f96015ca0105213523c',
'8a858f265c2187bd015c254c6c395bef',
'8a858f2c59cb2d550159cbf3d0437cb5',
'8a858f265c2187bd015c2686b0ad1782',
'8a858f7a5b39a003015b39ca67ab2efb',
'8a858ece5cd05041015cd19d339000b9',
'8a858e245b680423015b6fe6e6976937',
'8a858eeb5bb559d3015bb57fa13f10de',
'8a858fc959bb192f0159bb468bac1d4e',
'8a858f485d182cde015d186e2f753133',
'8a85892854a059ed0154aa7276e55a23',
'8a858ff259d1429e0159d2d0194c4b8e',
'8a8589f853f100fb01540ed945a70efd',
'8a858e7c5b15cf3b015b16ca44495b40',
'8a858ffa5ad810ff015ad82004960681',
'8a858f285c7d70a4015c7dd9108348aa',
'8a858f535bf336ac015bf721c64b7bd6',
'8a858fce5bf8129c015bfbcdde9467f6',
'8a858ed45bc9a645015bcd35a32e7c91',
'8a858f3b5c1c5a08015c1c6897ac062b',
'8a858fab5b9105dc015b999726de4412',
'8a858fea5b391cd7015b393aed9e2a04',
'8a858fb85b81c658015b853e87955e7f',
'8a858fec5b5c994b015b5c9ababf013b',
'8a858f5e5cb23c77015cc061b752625b',
'8a858e985c110418015c112e311e1355',
'8a858fb959976ce0015998f0a628583b',
'8a858e105bd92644015bdd2f7a981936',
'8a858f6c5bf84cd0015bf867a6ed0981',
'8a858ebb5adce699015adea5d5ae51b3',
'8a858e325b3e903a015b3e9a2fba0b3e',
'8a8589d8548faed501549a0fd1d72170',
'8a858e1d5cd58f9e015cdfe7c50d380b',
'8a858e1556321b940156368d95d74291',
'8a858fc55b2fde30015b300d85e20ef2',
'8a858f605b911d60015b9170209c2fb2',
'8a858e51584a1fc701584ef9749a7d80',

'8a85898a53bc15600153cdd9fae978e0',
'8a858f725b49c0c0015b525cb85f0552',
'8a858ee45b39aa80015b3b1a1d600f18',
'8a858ef458d4a52e0158d959e7501c2f',
'8a858f385b34d285015b388b44a27edf',
'8a858e9d5bfd7037015c07cdd6f1278d',
'8a858f375b493337015b494169b00e07',
'8a858eb75c21a2b9015c2a3cd3363330',
'8a858e055ac84a51015ac944e21e6883',
'8a858f385b34d285015b387bcb86740a',
'8a858ebd5b254c8b015b254cc8580023',
'8a858fb65b2020b2015b23c6042023b2',
'8a858f1d5c7db9e2015c7dcd87810cd3',
'8a858e245b680423015b7ad113d6259a',
'8a858f085cf4e173015cf88e44cb69e4',
'8a858e345bd96da8015bda7ce5f33ec9',
'8a858f545ad7fa6f015adbe5c20e17bc',
'8a858f335b61adff015b61b479ea0494',
'8a858eba5c884d2a015c9158d0692e5b',
'8a858ec65cc6352b015cc75880264271',
'8a76e7d443e6e97c0143ed0a33375981',
'8a858e675c3fe0a1015c549c328835a8',
'8a858fa25bd92c20015be3783b7a6a29',
'8a858f7d5cf9aef9015cffc97211089d',
'8a858fa75b5c94ed015b5ca7267d13f2',
'8a858f975c4582c4015c561116aa65ea',
'8a858f565b683b56015b6e9302a77dd1',
'8a858e255a423087015a466009d66e93',
'8a858fc75b90e6b5015b9c1cebad3e18',
'8a858ece5c88b58b015c88d082290889',
'8a858ec559dbb41a0159dd81258b41c4',
'8a858e6a5bfd4b54015bfecb639d6d1a',
'8a858f735aec46c4015aec68410613ed',
'8a858e4357be1daf0157c4213f1968e4',
'8a858f985b3db23f015b3dbdadc8145b',
'8a858fd95c3b5bca015c438aac9e415e',
'8a858f70575165bf0157564b41f10c18',
'8a858f235b8bf5bc015b8bf94dbc0278',
'8a858ef255c55fd60155c738de611535',
'8a858fc55b2548dd015b2fd12423240e',
'8a85882653bc51f50153d00a923d26cb',
'8a858e51584a1fc701584a2d9653035b',
'8a858e585693982b0156adba96202a26',
'8a858e395cb1d4d9015cc15f03193c90',
'8a858899538ddb8e0153a780c56e34bb',
'8a858fcb5ca240f7015ca6166a3375c4',
'8a858f255ca276c5015ca6175ff526db',
'8a858e345c788b37015c792c68fd5717',
'8a858e5a5be99e1c015be9fc8d12143c',
'8a858f605b911d60015b9b0d0fce6913',
'8a858f255cb1710a015cb851bf7c77fe',
'8a858e6d5d13b3ca015d15eb12664fd9',
'8a858edc5c7e7da5015c7e8d8b120593',
'8a858f275c451af5015c5426a1de3649',
'8a858e9b560bbd6301561ea184a0102d',
'8a858f2b5bf35819015bf3cc59193622',
'8a858fef570f30e501571f8c9b393e0c',
'8a858ea555d9e6020155ddcfc66d05ec',
'8a858e695d1e075c015d2ffe8d7d6371',
'8a858e0256b781d80156ca9f1d3c2e72',
'8a858e145bf81459015bfb14c0187f03',
'8a858f285c7d70a4015c7d86438a0fa5',
'8a858efe5bca0a81015bcafa92f72787',
'8a858eba5c884d2a015c91faae80473d',
'8a858fdf5bf85f71015bf8bc66792023',
'8a858fcb5ca240f7015ca67284a42f30',
'8a8589045384bd8301539e134b745638',
'8a858eb75c21a2b9015c24d4d3203a06',
'8a858fad5bf85d5e015bf96d6d254a0d',
'8a858e875b910dfe015ba4a5adbe3ea5',
'8a858e6c5c88d145015c8bad507561b8',

```
'8a858e725c3ae262015c45243caa4f4d',
'8a858ff25c8250c1015c82907bc62b42',
'8a858f30551130db01552563f6780605',
'8a858fca5c830943015c870395876947',
'8a858f165bf7b5ca015bf7b8e30e0199',
'8a858e255b910db1015b91c0c8433384',
'8a858f995bcdb303015bcdb8f52f0478',
'8a858f99560bbe45015613f9b9fa2171',
'8a858e965b5e1b91015b617bcea5726c',
'8a858fdf5bf85f71015bf90560683095',
'8a858fa95c695f85015c780af573028d',
'8a858e135cb22031015cb723f07f66f8',
'8a858f7958b0a4a80158b4e352b6367b',
'8a858fec5c169ff0015c16a935bc04b2',
'8a858fb65b2020b2015b205fe7a63993',
'8a858ee65be8d42d015bec93bd4c5303',
'8a858e395cb1d4d9015cbd121e2a20c7',
'8a858e875c63d395015c6ef64f313cc2',
'8a858e625c8d993a015c9bf9278c32f7',
'8a858fa55cc5dbbc015cc5f1c8930e9a',
'8a858f7e5c886ca9015c89215212546b',
'8a858f1e5add5268015ade70d6e220fa',
'8a858eda5c8863ff015c9dead65807bb',
'8a858ec75c11a07a015c15742adc38b2',
'8a858f585bfd6341015c00cba7e11878',
'8a858f0455d9feaf0155ea840c8f410a',
'8a858e8f5d41c974015d48519bbf103f',
'8a858f4655ca643c0155ccc7c480502f',
'8a858ea75cef5535015cf36a4ef31265',
'8a858f4e5ca72981015ca78d16c126e2',
'8a858ea35c7ce42e015c7d14e11519d1',
'8a858eed5af07cc4015af0cc483126df',
'8a858e6f5cd5e874015ce084c86e2a87',
'8a858eb75c21a2b9015c34ec66d87d38',
'8a858eda5c8863ff015c927d4b997fa1',
'8a858ec75bfd77c4015c0cf0e67618d2',
'8a858eda5c8863ff015c96c523266b34',
'8a858e6f5668e01701567140f0f5212c',
'8a858f255ca276c5015ca63c859537ef',
'8a858e1f5cac899f015cad22e40430d2',
'8a858eba5b681df4015b7b3b976f62da',
'8a858e225c404292015c541632fc2d77',
'8a8589a453bc422d0153c7aef31f02bd',
'8a858e4f58b9f0b40158c027a53e65a8',
'8a858e885c63d379015c6780ca836514',
'8a858ed55c63db54015c69c1af982dbf',
'8a858f295ca6f581015ca77afccb5d6d',
'8a858ee55cb156e6015cb63f19146515',
'8a858e2d5bb55d46015bb55e67600058',
'8a858f2e5c699f3a015c6d9e1364767d',
'8a858f295c8d307f015c95b5b13724f7',
'8a858f0f5bfd79d2015c0c2370d436a1',
'8a858f8f5bfd3cfa015bfe947ef14c93',
'8a858f565b683b56015b70b951992639',
'8a858f435aa9712e015aa9e8e9735ca6',
'8a858fab5cd5e1a7015cd9c3b9ae0f29',
'8a858fd85b685607015b7bf160dd7dfe',
'8a858e105bd92644015bd92816fa0073',
'8a858fa95c695f85015c6aba473a5478',
'8a858f475be98d5f015be991bd340215',
'8a858fa45bd95e9a015bdc9f83f311fb',
'8a858f3e589c40610158b48a71b430d3',
'8a858f045bc9690c015bc9a82a3c19aa',
'8a858ff45be92ea2015be991525a2f79',
'8a858e005ca7abe3015cac3d9960218c',
'8a858f7d5cf9aef9015cfe8fde1763ac',
'8a858f0f5bfd79d2015c02f301117179',
'8a858e325d60aaf2015d61746b213f1f',
'8a858ffd5c8d343d015c9be64cc850f0',
'8a858ed1594a282f01594a4d14f50a58',
'8a858fa154e2912f0154ee239b0857f0',
```

```
       '8a858e285aa8cfc1015aa8fd90951f5a',
       '8a858ed95cc5bd6b015cc62436ae43be',
       '8a858fab5cd5e1a7015ce34b7abd0bb2',
       '8a858ffc5852f33d01585542e78e06f8',
       '8a858f795896e56b0158a4eb84d1648d',
       '8a858fce5bf8129c015bf9ff58e90318',
       '8a858f525bedff8d015bef815d3c6d46',
       '8a858fa15c8ceaaa015c8d7922144eda',
       '8a858f975c4582c4015c4e46f789665d',
       '8a858e9a54d9121b0154dd1bfff23870',
       '8a858fdf5bf85f71015bf93c8dd53cc7',
       '8a858fe756939bbb0156993f3dda2d08',
       '8a858e875b910dfe015b9ead5034701d',
       '8a858e045b495c30015b49b40d522144',
       '8a858f225b3dc49e015b3ddbb4e81dee',
       '8a858e005ca7abe3015ca840481d219b',
       '8a858f435cd01e6a015cd0bafa8d502f',
       '8a858fd45a4260bc015a437a12c853c0',
       '8a858f9f5bd99987015be41c167f470c',
       '8a858faf56939fea01569f7f791b4369',
       '8a858f5c5ad7d927015ad8d3ea7b5cca',
       '8a8589ec54517bf901546122c8862bab',
       '8a858ed45c454c11015c4e1307e35dd9',
       '8a858fcb5c87dd04015c882423cd3abd',
       '8a858e0f5c45466f015c4bf3a7cd77bc',
       '8a858e725c360240015c372f0894441b',
       '8a858eec5d4232a5015d4623cd0a048b',
       '8a858f3f5c35c74a015c38ba4efc74cb',
       '8a858ed45c454c11015c544fc1dd3dd4',
       '8a858eba5c884d2a015c93b1d62410fb',
       '8a858e935b496584015b496a6fde01f6',
       '8a858f365b2547f3015b307651737158',
       '8a858f255cb1710a015cb52df5760337',
       '8a858f055b15b8d3015b164758eb3a24',
       '8a858e1e5b47f961015b480939510c02',
       '8a81899e529ba94b01529c8baba42daa',
       '8a858ea55ac1546c015ac328fdff18ed',
       '8a8189f9528494e4015284e68a250edb',
       '8a858f8f5bfd3cfa015bff332a9455b1',
       '8a858e4c5a715421015a7160ed750369',
       '8a858ea45b35639b015b377182544104',
       '8a858e345bd96da8015bd97f85df06c8',
       '8a858f085b905684015b9078144f1fcc',
       '8a858ee55cb156e6015cb6a5d8407bd1',
       '8a858e785c8d9167015c9341c238485a',
       '8a858f995c63d7ef015c65a8d83a03c8',
       '8a858e435bedeb4f015beefb8a294d30',
       '8a858eee58727169015873e03f24718a',
       '8a858f335c8d9723015c9904394f186f',
       '8a858e2155defefd0155e88ca1b22485',
       '8a858eb75c21a2b9015c29ebece12d01',
       '8a858ff15b5d3f84015b5d509bd61cae'], dtype=object)
```

The referredby column has a high cardinality. The details of the referee may have been useful though but due to the large number of missing values that cannot easily be imputed, the column value cannot particularly give any relevance to the model. This feature will not be used.

first_due_date and first_repaid_date columns

This columns individually don't provide significant features but when used together might prove to be more useful. The difference between the two columns can tell us if a customer was prompt on making their payments.

In [137…

```
#isolating the firstduedate, firstrepaiddate and good_bad_flag into a dataframe of its own
loanrepayment = previousCustomers[['firstduedate','firstrepaiddate','good_bad_flag']]
```

In [137…

```
loanrepayment.head()
```

Out[1373]:

	firstduedate	firstrepaiddate	good_bad_flag
0	2016-09-14 00:00:00.000000	2016-09-01 15:51:43.000000	Good
1	2017-05-30 00:00:00.000000	2017-05-26 00:00:00.000000	Good
2	2017-04-04 00:00:00.000000	2017-04-26 22:03:47.000000	Good
3	2017-07-03 00:00:00.000000	2017-06-25 15:13:56.000000	Good
4	2017-07-31 00:00:00.000000	2017-07-25 08:04:27.000000	Good

In [137…

```
loanrepayment['firstduedate'] = pd.to_datetime(loanrepayment['firstduedate'])
loanrepayment['firstrepaiddate'] = pd.to_datetime(loanrepayment['firstrepaiddate'])
```

In [137…

```
# the difference between the closedDate and the creationDate is stored in a new column
# called 'firstrepaymentlapse'
loanrepayment['firstrepaymentlapse'] = ((loanrepayment.firstduedate - loanrepayment.firstrepaiddate)/np.timedelta64(1, 'D'))
loanrepayment.head()
```

Out[1375]:

	firstduedate	firstrepaiddate	good_bad_flag	firstrepaymentlapse
0	2016-09-14	2016-09-01 15:51:43	Good	12.339086
1	2017-05-30	2017-05-26 00:00:00	Good	4.000000
2	2017-04-04	2017-04-26 22:03:47	Good	-22.919294
3	2017-07-03	2017-06-25 15:13:56	Good	7.365324
4	2017-07-31	2017-07-25 08:04:27	Good	5.663576

For the column 'firstrepaymentlapse', the positive values indicate someone who was able to pay before the due date and negative values are someone who paid after the due date

Updating selected features:

1. period take to settle the loan(lifetime of loan)
2. first repayment day status: loanrepayment

The next cells are looking at the data from the performance dataframe. Those columns that were also present in the previous dataframe and were dropped as features above, will also be dropped as features below.

perf_referredby

In [137…

```
len(previousCustomers['perf_referredby'].unique())
```

Out[1376]:

```
408
```

In [137…

```
previousCustomers['perf_referredby'].isnull().sum()
```

Out[1377]:

```
12873
```

There are a number of null values within this columns and we cannot easily replace the missing values. This column will therefore not be used going forward

birthdate

This column may not be significant as it is for training our model. It may be better to get the actual age of the customers rather than using there birthdates.

So let us get the actual ages of the customers.

In [137…

```
#converting birthdate column to datetime
previousCustomers['birthdate'] = pd.to_datetime(previousCustomers['birthdate'])
```

In [137…

```
customerAge = previousCustomers[['birthdate','good_bad_flag']]
customerAge.head()
```

Out[1379]:

	birthdate	good_bad_flag
0	1972-01-15	Good
1	1972-01-15	Good
2	1972-01-15	Good
3	1972-01-15	Good
4	1972-01-15	Good

In [138…
```
currentDate = date.today()
```

In [138…
```
currentDate
```

Out[1381]:
```
datetime.date(2022, 12, 13)
```

In [138…
```
customerAge['age'] = round(((pd.datetime.now()- customerAge['birthdate'])/np.timedelta64(1, 'Y')),1)
```

In [138…
```
customerAge.tail()
```

Out[1383]:

	birthdate	good_bad_flag	age
13668	1986-02-20	Good	36.8
13669	1979-04-18	Good	43.7
13670	1989-11-19	Good	33.1
13671	1980-11-12	Good	42.1
13672	1989-10-24	Good	33.1

for the next graph, it was generated in Microsoft Excel. A pivot table was used to group the ages and get the number of bad loans in each age group, it was then displayed in the people graph. The files being saved were exported to excel

In [138…
```
customerAge.to_csv('allLoansAndAges.csv')
```

alt text

alt text

so we shall use the ages of people to train our model instead of their birthdate

Updating selected features:

1. period take to settle the loan(lifetime of loan)
2. first repayment day status: loanrepayment
3. ages of the customers

bank account type: type of primary bank account

In [138…
```
len(previousCustomers['bank_account_type'].unique())
```

Out[1385]:
```
3
```

In [138…
```
testdemo = pd.read_csv('testdemographics.csv')
```

In [138…
```
len(testdemo['bank_account_type'].unique())
```

Out[1387]:
```
3
```

In [138…
```
previousCustomers['bank_account_type'].isnull().sum()
```

Out[1388]:
```
0
```

In [138…
```
previousCustomers['bank_account_type'].unique()
```
Out[1389]:
```
array(['Other', 'Savings', 'Current'], dtype=object)
```

This feature shall be used to train our model

Updating selected features:

1. period take to settle the loan(lifetime of loan): previousCustomers
2. first repayment day status: loanrepayment
3. ages of the customers
4. the type of bank account

bank_name_clients: name of the bank

In [139…
```
previousCustomers['bank_name_clients'].unique()
```
Out[1390]:
```
array(['Diamond Bank', 'EcoBank', 'First Bank', 'GT Bank', 'UBA',
       'Union Bank', 'FCMB', 'Access Bank', 'Zenith Bank',
       'Fidelity Bank', 'Stanbic IBTC', 'Skye Bank', 'Sterling Bank',
       'Wema Bank', 'Keystone Bank', 'Unity Bank', 'Heritage Bank',
       'Standard Chartered'], dtype=object)
```

In [139…
```
len(previousCustomers['bank_name_clients'].unique())
```
Out[1391]:
```
18
```

In [139…
```
previousCustomers['bank_name_clients'].isnull().sum()
```
Out[1392]:
```
0
```

In [139…
```
len(testdemo['bank_name_clients'].unique())
```
Out[1393]:
```
18
```

This column will be used to train the model

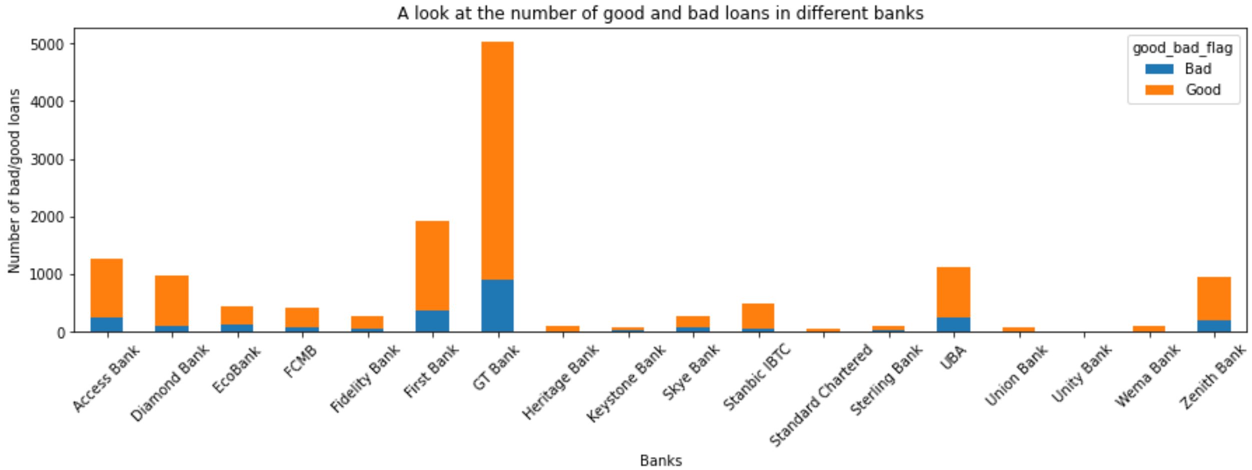
Updating selected features:

1. period take to settle the loan(lifetime of loan)
2. first repayment day status:
3. ages of the customers
4. the type of bank account
5. the bank name of the clients

We can also have a look the bad loans and good loans according to the different banks

In [139…
```
bankStats = previousCustomers[['bank_name_clients','good_bad_flag']]
```

In [139…
```
bankStats.groupby(['bank_name_clients','good_bad_flag']).size().unstack().plot(kind='bar',stacked=True,rot=45)
plt.xlabel('Banks')
plt.ylabel('Number of bad/good loans')

plt.title('A look at the number of good and bad loans in different banks')

f = plt.gcf()
f.set_figwidth(15)
```



From the graph above, we can see that the highest number of loans, and subsequently the highest number of bad loans, taken out has been from GT bank.

In [139…
```
# FIND THE AGE RANGE OF THE PEOPLE WHO ARE GOING TO THE DIFFERENT BANKS,
#use pivot table and see the most common bank in the
#age range and see if there's a connection
```

bank branch clients

In [139…
```
previousCustomers['bank_branch_clients'].unique()
```

Out[1397]:
```
array([nan, 'OBA ADEBIMPE', 'RING ROAD', 'AKUTE', 'OGBA',
       'ADEOLA HOPEWELL', 'ABEOKUTA', 'OJUELEGBA', 'LAGOS',
       'OBA AKRAN BERGER PAINT',
       'ACCESS BANK PLC, CHALLENGE ROUNDABOUT IBADAN, OYO STATE.',
       'BOSSO ROAD, MINNA',
       'PLOT 999C DANMOLE STREET, ADEOLA ODEKU, VICTORIA ISLAND, LAGOS',
       'MAFOLUKU', '17, SANUSI FAFUNWA STREET, VICTORIA ISLAND, LAGOS',
       'TRANS AMADI', 'APAPA', 'MUSHIN BRANCH', 'OAU ILE IFE',
       ' IDI - ORO MUSHIN', 'AJOSE ADEOGUN', 'TINCAN', 'ABULE EGBA',
       'OBA AKRAN', 'STERLING BANK PLC 102, IJU ROAD, IFAKO BRANCH',
       'LEKKI EPE', 'OGUDU, OJOTA', 'AKURE BRANCH',
       '40,SAPELE ROAD ,OPPOSITE DUMAZ JUNCTION BENIN CITY EDO STATE.'],
      dtype=object)
```

In [139…
```
previousCustomers['bank_branch_clients'].isnull().sum()
```

Out[1398]:
```
13569
```

This column will not be used to train our model due to the large number of missing values that cannot easily be replaced.

employment status clients

In [139…
```
previousCustomers['employment_status_clients'].unique()
```

Out[1399]:
```
array(['Permanent', nan, 'Self-Employed', 'Student', 'Unemployed',
       'Retired', 'Contract'], dtype=object)
```

In [140…
```
#loading the testdemographics dataset to see if the values in the previous cell
#are also the values in the test dataset
testdemo = pd.read_csv('testdemographics.csv')
```

In [140…
```
testdemo['employment_status_clients'].unique()
```

Out[1401]:
```
array(['Permanent', 'Self-Employed', nan, 'Student', 'Contract',
       'Unemployed', 'Retired'], dtype=object)
```

In [140…
```
previousCustomers['employment_status_clients'].isnull().sum()
```

Out[1402]: 1363

In [140…
```
#imputing the missing values with 'Not given'
previousCustomers['employment_status_clients'].fillna("Employment_not-given", inplace=True)
```

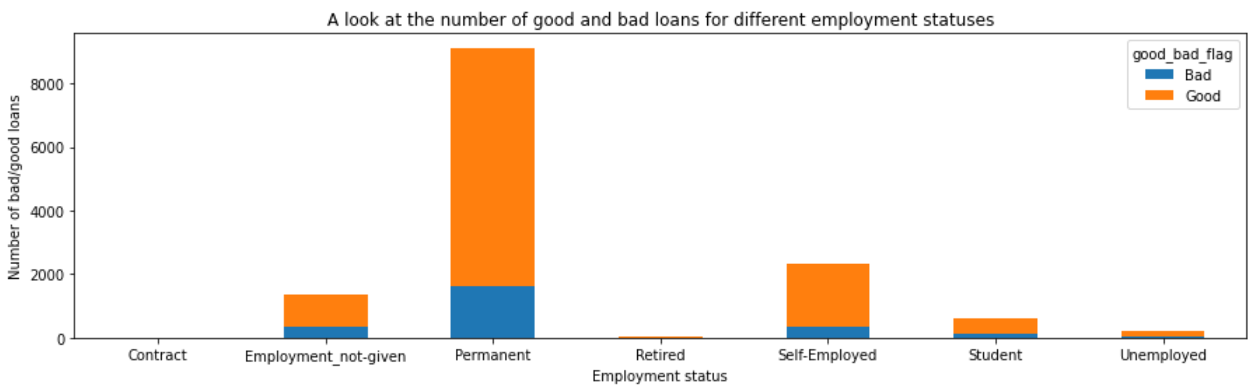
Updating selected features:

1. period take to settle the loan(lifetime of loan): previousCustomers[loanlifewithdelta]
2. first repayment day status: loanrepayment[firstrepaymentlapsewithdelta]
3. ages of the customers: birthdate[age]
4. the type of bank account: previousCustomers[bank_account_type]
5. the bank name of the clients: previousCustomers[bank_name_clients]
6. the employment status of clients: previousCustomers[employment_status]

We can also see how the number of bad and good loans relates to a person's employment status

In [140…
```
jobstatus = previousCustomers[['employment_status_clients','good_bad_flag']]
```

In [140…
```
jobstatus.groupby(['employment_status_clients','good_bad_flag']).size().unstack().plot(kind='bar',stacked=True,rot=0)
plt.xlabel('Employment status')
plt.ylabel('Number of bad/good loans')

plt.title('A look at the number of good and bad loans for different employment statuses')

#plt.subplot(4,1,2)
#plt.title('A look at the number of good and bad loans throughout the years')

f = plt.gcf()
f.set_figwidth(15)
```



From the graph above, we can see that people of 'Permanent' employment type take out the most loans. They also have the most bad loans as compared to other employment statuses.

level of education of clients

In [140…
```
previousCustomers['level_of_education_clients'].unique()
```

Out[1406]:
```
array(['Post-Graduate', nan, 'Primary', 'Graduate', 'Secondary'],
      dtype=object)
```

In [140…
```
previousCustomers['level_of_education_clients'].isnull().sum()
```

Out[1407]: 10209

There is a high number of missing values but this column can be a good feature to train our model on. The missing values will therefore be imputed with 'Not-given'

In [140…
```
previousCustomers['level_of_education_clients'].fillna('Education_not-given', inplace=True)
```

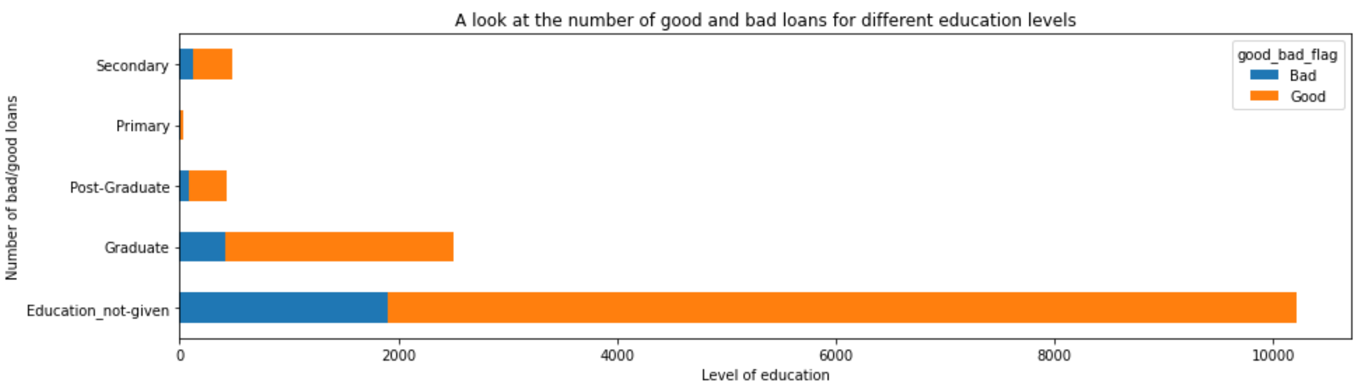
Updating selected features:

1. period take to settle the loan(lifetime of loan): previousCustomers[loanlifewithdelta]
2. first repayment day status: loanrepayment[firstrepaymentlapsewithdelta]
3. ages of the customers: birthdate[age]
4. the type of bank account: previousCustomers[bank_account_type]
5. the bank name of the clients: previousCustomers[bank_name_clients]
6. the employment status of clients: previousCustomers[employment_status]
7. the level of education of clients: previousCustomers[level_of_education_clients]

We can still look at the how the level of education plays out in the matter of bad and good loans

In [140…
```
educationlevel = previousCustomers[['level_of_education_clients','good_bad_flag']]
```

In [141…
```
#plt.subplot(4,1,1)
#yearLook = closedDates[['year', 'good_bad_flag']]
educationlevel.groupby(['level_of_education_clients','good_bad_flag']).size().unstack().plot(kind='barh',stacked=True,rot=0)
plt.xlabel('Level of education')
plt.ylabel('Number of bad/good loans')

plt.title('A look at the number of good and bad loans for different education levels')

#plt.subplot(4,1,2)
#plt.title('A look at the number of good and bad loans throughout the years')

f = plt.gcf()
f.set_figwidth(15)
```



We can see that the people whose education level we don't know take out the highest number of loans and also have the highest number of bad loans. This is then followed by the graduate education level.

So we have gone through the non-numerical features of our data and have selected the following 6 columns as features to train our model with:

1. period take to settle the loan(lifetime of loan)
2. first repayment day status
3. ages of the customers
4. the type of bank account:
5. the bank name of the clients
6. the level of education of clients
7. the employment status of clients

Now to look at the numerical columns

In [141…
```
numerical = [var for var in previousCustomers.columns if previousCustomers[var].dtype!='O']

print('There are {} numerical variabes \n'.format(len(numerical)))
```

```
print('They are: ', numerical)
```

```
There are 14 numerical variabes

They are:  ['loannumber', 'creationdate', 'loanamount', 'totaldue', 'termdays', 'closeddate', 'perf_loannumber', 'perf_loanamount', 'perf_totaldue', 'perf_termdays', 'birthdate', 'longitude_gps', 'latitude_gps', 'loanlife']
```

We are going to ignore creationdate, closeddate, birthdate, loanlife as these have been dealt with and discussed while looking at the categorical features.

In [141...
```
numColumns= previousCustomers[['loannumber','loanamount','totaldue','termdays','closeddate','perf_loannumber','perf_loanamount','perf_totaldue','perf_termdays','longitude_gps','latitude_gps']]
```

In [141...
```
numColumns.head()
```

Out[1413]:

	loannumber	loanamount	totaldue	termdays	closeddate	perf_loannumber	perf_loanamount	perf_totaldue	perf_termdays	longitude_gps	latitude_gps
0	2	10000.0	13000.0	30	2016-09-01 16:06:48	12	30000.0	34500.0	30	3.43201	6.433055
1	9	10000.0	13000.0	30	2017-05-28 14:44:49	12	30000.0	34500.0	30	3.43201	6.433055
2	8	20000.0	23800.0	30	2017-04-26 22:18:56	12	30000.0	34500.0	30	3.43201	6.433055
3	10	20000.0	24500.0	30	2017-06-25 15:24:06	12	30000.0	34500.0	30	3.43201	6.433055
4	11	20000.0	24500.0	30	2017-07-25 08:14:36	12	30000.0	34500.0	30	3.43201	6.433055

Below we add some of the numerical columns generated from the categorical features. These added columns are going to be treated as numerical i.e a person's age, the lifetime of a loan, the time a person took to complete the first repayment day.

In [141...
```
numColumns = pd.concat([numColumns, customerAge['age'],previousCustomers['loanlife'],loanrepayment['firstrepaymentlapse'],previousCustomers['good_bad_flag']],axis=1)
```

In [141...
```
numColumns.tail()
```

Out[1415]:

	loannumber	loanamount	totaldue	termdays	closeddate	perf_loannumber	perf_loanamount	perf_totaldue	perf_termdays	longitude_gps	latitude_gps	age	loanlife	firstrepaymentlapse	good_bad_flag
13668	1	10000.0	11500.0	15	2017-07-18 16:33:55	2	10000.0	11500.0	15	5.252457	12.991440	36.8	2.231736	12.316840	Good
13669	1	10000.0	13000.0	30	2017-07-11 14:26:40	2	10000.0	13000.0	30	7.478858	9.055714	43.7	27.359097	2.405197	Good
13670	1	10000.0	11500.0	15	2017-06-26 14:02:03	2	10000.0	11500.0	15	3.381677	6.455923	33.1	9.865671	6.422292	Good
13671	1	10000.0	11500.0	15	2017-07-05 14:31:17	2	10000.0	13000.0	30	6.979660	4.879515	42.1	6.587141	8.401991	Good
13672	1	10000.0	11500.0	15	2017-02-15 09:06:34	2	10000.0	13000.0	30	7.530892	9.042928	33.1	26.837870	-12.369039	Good

In [141...
```
numColumns.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13673 entries, 0 to 13672
Data columns (total 15 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   loannumber           13673 non-null  int64
 1   loanamount           13673 non-null  float64
 2   totaldue             13673 non-null  float64
 3   termdays             13673 non-null  int64
 4   closeddate           13673 non-null  datetime64[ns]
 5   perf_loannumber      13673 non-null  int64
 6   perf_loanamount      13673 non-null  float64
 7   perf_totaldue        13673 non-null  float64
 8   perf_termdays        13673 non-null  int64
 9   longitude_gps        13673 non-null  float64
 10  latitude_gps         13673 non-null  float64
 11  age                  13673 non-null  float64
 12  loanlife             13673 non-null  float64
 13  firstrepaymentlapse  13673 non-null  float64
 14  good_bad_flag        13673 non-null  object
dtypes: datetime64[ns](1), float64(9), int64(4), object(1)
memory usage: 1.7+ MB
```

In [141…

```python
#dropping the closeddate column as it's not needed
numColumns.drop(['closeddate'],axis=1,inplace=True)
```

From the first repayment lapse column, we are going to create two new columns:

1. The first, LateFirstPay, will show whether someone was late in making the first repayment date and by how many days
2. The second, EarlyFirstPay, will show whether someone was early in making the first repayment date and by how many days

In [141…

```python
numColumns.loc[numColumns['firstrepaymentlapse'] < 0, 'LateFirstPay'] = numColumns['firstrepaymentlapse']
numColumns.loc[numColumns['firstrepaymentlapse'] < 0, 'EarlyFirstPay'] = 0
numColumns.loc[numColumns['firstrepaymentlapse'] >= 0, 'LateFirstPay'] = 0
numColumns.loc[numColumns['firstrepaymentlapse'] >= 0, 'EarlyFirstPay'] = numColumns['firstrepaymentlapse']
```

In [141…

```python
numColumns
```

Out[1419]:

	loannumber	loanamount	totaldue	termdays	perf_loannumber	perf_loanamount	perf_totaldue	perf_termdays	longitude_gps	latitude_gps	age	loanlife	firstrepaymentlapse	good_bad_flag	LateFirstPay	EarlyFirstP
0	2	10000.0	13000.0	30	12	30000.0	34500.0	30	3.432010	6.433055	50.9	16.947407	12.339086	Good	0.000000	12.3390
1	9	10000.0	13000.0	30	12	30000.0	34500.0	30	3.432010	6.433055	50.9	29.879120	4.000000	Good	0.000000	4.0000
2	8	20000.0	23800.0	30	12	30000.0	34500.0	30	3.432010	6.433055	50.9	52.515706	-22.919294	Good	-22.919294	0.0000
3	10	20000.0	24500.0	30	12	30000.0	34500.0	30	3.432010	6.433055	50.9	24.117882	7.365324	Good	0.000000	7.3653
4	11	20000.0	24500.0	30	12	30000.0	34500.0	30	3.432010	6.433055	50.9	26.927905	5.663576	Good	0.000000	5.6635
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
13668	1	10000.0	11500.0	15	2	10000.0	11500.0	15	5.252457	12.991440	36.8	2.231736	12.316840	Good	0.000000	12.3168
13669	1	10000.0	13000.0	30	2	10000.0	13000.0	30	7.478858	9.055714	43.7	27.359097	2.405197	Good	0.000000	2.4051
13670	1	10000.0	11500.0	15	2	10000.0	11500.0	15	3.381677	6.455923	33.1	9.865671	6.422292	Good	0.000000	6.4222
13671	1	10000.0	11500.0	15	2	10000.0	13000.0	30	6.979660	4.879515	42.1	6.587141	8.401991	Good	0.000000	8.4019
13672	1	10000.0	11500.0	15	2	10000.0	13000.0	30	7.530892	9.042928	33.1	26.837870	-12.369039	Good	-12.369039	0.0000

13673 rows × 16 columns

In [142…

```python
numColumns.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13673 entries, 0 to 13672
Data columns (total 16 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   loannumber           13673 non-null  int64
 1   loanamount           13673 non-null  float64
 2   totaldue             13673 non-null  float64
 3   termdays             13673 non-null  int64
 4   perf_loannumber      13673 non-null  int64
 5   perf_loanamount      13673 non-null  float64
 6   perf_totaldue        13673 non-null  float64
 7   perf_termdays        13673 non-null  int64
 8   longitude_gps        13673 non-null  float64
 9   latitude_gps         13673 non-null  float64
 10  age                  13673 non-null  float64
 11  loanlife             13673 non-null  float64
 12  firstrepaymentlapse  13673 non-null  float64
 13  good_bad_flag        13673 non-null  object
 14  LateFirstPay         13673 non-null  float64
 15  EarlyFirstPay        13673 non-null  float64
dtypes: float64(11), int64(4), object(1)
memory usage: 2.3+ MB
```

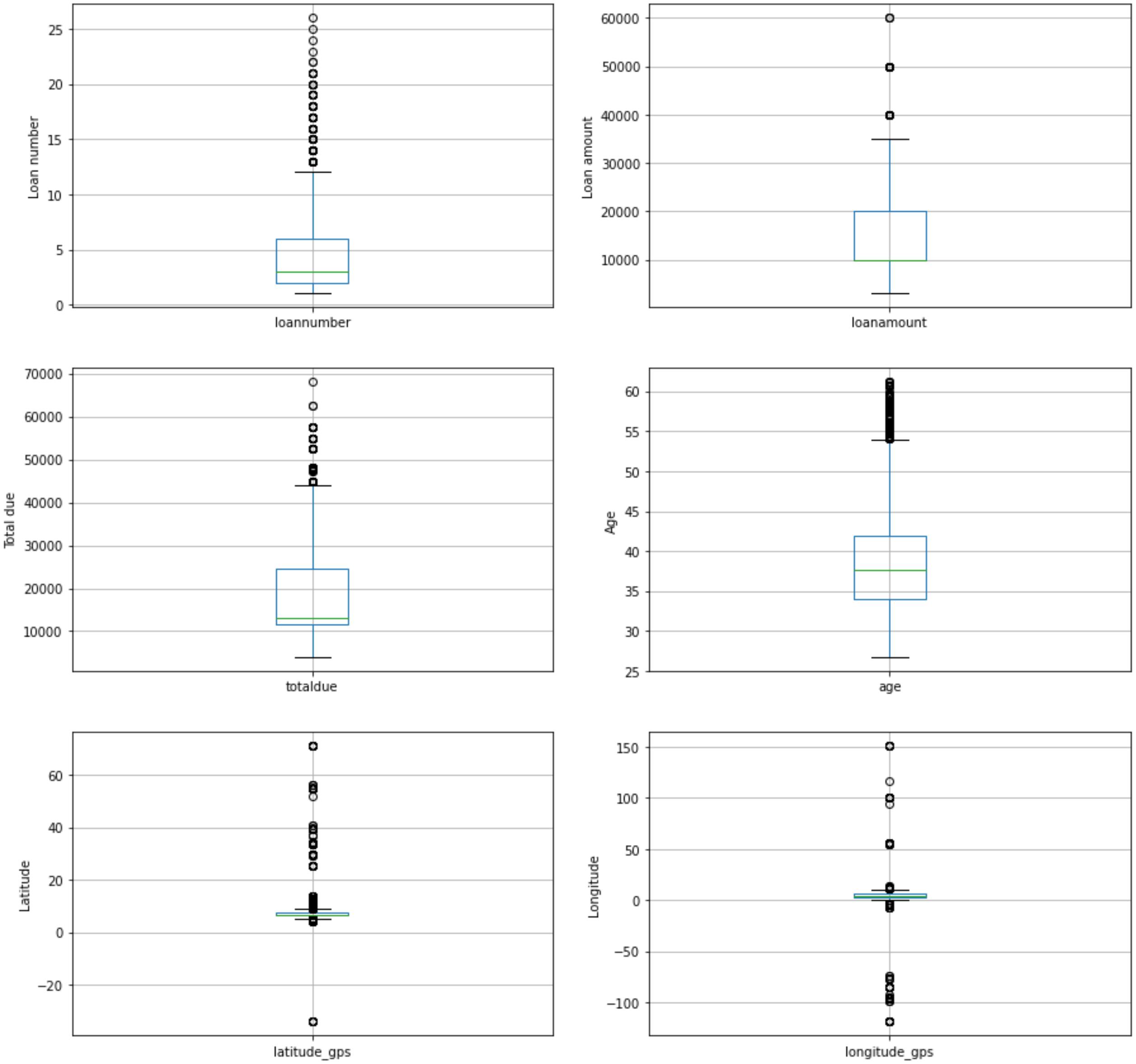
In [142…

```python
#dropping the firstrepaymentlapse column in favour of the two new ones created
numColumns.drop(['firstrepaymentlapse'], axis=1, inplace=True)
```

Let's have a look at if there are outliers in our data using boxplot. The term_loan column will not be checked since it has only 4 values.

In [142…

```
plt.figure(figsize=(15,10))

plt.subplot(3,2,1)
fig = numColumns.boxplot(column='loannumber')
fig.set_ylabel('Loan number')

plt.subplot(3,2,2)
fig = numColumns.boxplot(column='loanamount')
fig.set_ylabel('Loan amount')

plt.subplot(3,2,3)
fig = numColumns.boxplot(column='totaldue')
fig.set_ylabel('Total due')

plt.subplot(3,2,4)
fig = numColumns.boxplot(column='age')
fig.set_ylabel('Age')

plt.subplot(3,2,5)
fig = numColumns.boxplot(column='latitude_gps')
fig.set_ylabel('Latitude')

plt.subplot(3,2,6)
fig = numColumns.boxplot(column='longitude_gps')
fig.set_ylabel('Longitude')

f = plt.gcf()
f.set_figheight(15)
```



For our numerical columns, we can see that there are outliers in each column.

For the loanamount and totaldue, this has to do with amounts of money and in such a classification problem, we believe these outliers are important as the greater amount may imply the occurrence of a bad loan.

For the age column, these outliers don't need to be dropped, just showing an age group(54-around 65) who are not usually loan clients.

The outliers via longitude and latitude may be atttibuted to people who are outside Nigeria. And this may also act as an indicator if people from outside the country are more like to have bad loans or not.

For loan number, we can see what kind of data is contained there.

loannumber: The number of the loan that you have to predict

In [150…

```
numColumns['loannumber'].value_counts()
```

Out[1501]:

```
1     3255
2     2228
3     1730
4     1405
5     1158
6      967
7      778
8      614
9      475
10     333
11     255
12     169
13     110
14      73
15      40
16      26
17      17
18      14
19       8
20       6
21       6
22       2
24       1
23       1
25       1
26       1
Name: loannumber, dtype: int64
```

For the loan number column, it is not a column with more information to help determine if the outliers are bad or good. But from the column description, it seems significant as a whole and will not be dropped.

So all these numerical columns will be kept.

splitting data into target and feature

In [142…

```
#creating a dataframe consisting of all of the columns that are going to be used to train
#our models:all the numerical columns, bank account type, name of the bank, employment
#status and level of education
newdf = pd.concat([numColumns,
                   pd.get_dummies(previousCustomers.bank_account_type),
                   pd.get_dummies(previousCustomers.bank_name_clients),
                   pd.get_dummies(previousCustomers.employment_status_clients),
                   pd.get_dummies(previousCustomers.level_of_education_clients)], axis=1)

newdf
```

Out[1423]:

	loannumber	loanamount	totaldue	termdays	perf_loannumber	perf_loanamount	perf_totaldue	perf_termdays	longitude_gps	latitude_gps	...	Permanent	Retired	Self-Employed	Student	Unemployed	Education_not-given	G
0	2	10000.0	13000.0	30	12	30000.0	34500.0	30	3.432010	6.433055	...	1	0	0	0	0	0	
1	9	10000.0	13000.0	30	12	30000.0	34500.0	30	3.432010	6.433055	...	1	0	0	0	0	0	
2	8	20000.0	23800.0	30	12	30000.0	34500.0	30	3.432010	6.433055	...	1	0	0	0	0	0	
3	10	20000.0	24500.0	30	12	30000.0	34500.0	30	3.432010	6.433055	...	1	0	0	0	0	0	
4	11	20000.0	24500.0	30	12	30000.0	34500.0	30	3.432010	6.433055	...	1	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
13668	1	10000.0	11500.0	15	2	10000.0	11500.0	15	5.252457	12.991440	...	1	0	0	0	0	1	
13669	1	10000.0	13000.0	30	2	10000.0	13000.0	30	7.478858	9.055714	...	1	0	0	0	0	1	
13670	1	10000.0	11500.0	15	2	10000.0	11500.0	15	3.381677	6.455923	...	1	0	0	0	0	1	
13671	1	10000.0	11500.0	15	2	10000.0	13000.0	30	6.979660	4.879515	...	0	0	0	0	1	0	
13672	1	10000.0	11500.0	15	2	10000.0	13000.0	30	7.530892	9.042928	...	1	0	0	0	0	1	

13673 rows × 48 columns

In [142…
```
newdf.columns
```

Out[1424]:
```
Index(['loannumber', 'loanamount', 'totaldue', 'termdays', 'perf_loannumber',
       'perf_loanamount', 'perf_totaldue', 'perf_termdays', 'longitude_gps',
       'latitude_gps', 'age', 'loanlife', 'good_bad_flag', 'LateFirstPay',
       'EarlyFirstPay', 'Current', 'Other', 'Savings', 'Access Bank',
       'Diamond Bank', 'EcoBank', 'FCMB', 'Fidelity Bank', 'First Bank',
       'GT Bank', 'Heritage Bank', 'Keystone Bank', 'Skye Bank',
       'Stanbic IBTC', 'Standard Chartered', 'Sterling Bank', 'UBA',
       'Union Bank', 'Unity Bank', 'Wema Bank', 'Zenith Bank', 'Contract',
       'Employment_not-given', 'Permanent', 'Retired', 'Self-Employed',
       'Student', 'Unemployed', 'Education_not-given', 'Graduate',
       'Post-Graduate', 'Primary', 'Secondary'],
      dtype='object')
```

In [142…
```
#separating the input features, X and the target feature, y that we want to predict
X = newdf.drop(['good_bad_flag'], axis=1)
y = newdf['good_bad_flag']
```

In [150…
```
newdf['good_bad_flag'].value_counts()
```

Out[1506]:
```
Good    11146
Bad      2527
Name: good_bad_flag, dtype: int64
```

In [142…
```
#Giving ourselves a test set of 20% of the initial records
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state= 4)
```

In [142…
```
#Looking at the shape of our training and testing data
#Training data has 10,938 entries and the testing data has 2,735 entries
X_train.shape, X_test.shape
```

Out[1427]:
```
((10938, 47), (2735, 47))
```

In [142…
```
X_train.columns
```

Out[1428]:
```
Index(['loannumber', 'loanamount', 'totaldue', 'termdays', 'perf_loannumber',
       'perf_loanamount', 'perf_totaldue', 'perf_termdays', 'longitude_gps',
       'latitude_gps', 'age', 'loanlife', 'LateFirstPay', 'EarlyFirstPay',
       'Current', 'Other', 'Savings', 'Access Bank', 'Diamond Bank', 'EcoBank',
       'FCMB', 'Fidelity Bank', 'First Bank', 'GT Bank', 'Heritage Bank',
       'Keystone Bank', 'Skye Bank', 'Stanbic IBTC', 'Standard Chartered',
       'Sterling Bank', 'UBA', 'Union Bank', 'Unity Bank', 'Wema Bank',
       'Zenith Bank', 'Contract', 'Employment_not-given', 'Permanent',
       'Retired', 'Self-Employed', 'Student', 'Unemployed',
       'Education_not-given', 'Graduate', 'Post-Graduate', 'Primary',
       'Secondary'],
      dtype='object')
```

In [142…
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10938 entries, 2894 to 1146
Data columns (total 47 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   loannumber           10938 non-null  int64  
 1   loanamount           10938 non-null  float64
 2   totaldue             10938 non-null  float64
 3   termdays             10938 non-null  int64  
 4   perf_loannumber      10938 non-null  int64  
 5   perf_loanamount      10938 non-null  float64
 6   perf_totaldue        10938 non-null  float64
 7   perf_termdays        10938 non-null  int64  
 8   longitude_gps        10938 non-null  float64
 9   latitude_gps         10938 non-null  float64
 10  age                  10938 non-null  float64
 11  loanlife             10938 non-null  float64
 12  LateFirstPay         10938 non-null  float64
 13  EarlyFirstPay        10938 non-null  float64
 14  Current              10938 non-null  uint8  
 15  Other                10938 non-null  uint8  
 16  Savings              10938 non-null  uint8  
 17  Access Bank          10938 non-null  uint8  
 18  Diamond Bank         10938 non-null  uint8  
 19  EcoBank              10938 non-null  uint8  
 20  FCMB                 10938 non-null  uint8  
 21  Fidelity Bank        10938 non-null  uint8  
 22  First Bank           10938 non-null  uint8  
 23  GT Bank              10938 non-null  uint8  
 24  Heritage Bank        10938 non-null  uint8  
 25  Keystone Bank        10938 non-null  uint8  
 26  Skye Bank            10938 non-null  uint8  
 27  Stanbic IBTC         10938 non-null  uint8  
 28  Standard Chartered   10938 non-null  uint8  
 29  Sterling Bank        10938 non-null  uint8  
 30  UBA                  10938 non-null  uint8  
 31  Union Bank           10938 non-null  uint8  
 32  Unity Bank           10938 non-null  uint8  
 33  Wema Bank            10938 non-null  uint8  
 34  Zenith Bank          10938 non-null  uint8  
 35  Contract             10938 non-null  uint8  
 36  Employment_not-given  10938 non-null  uint8  
 37  Permanent            10938 non-null  uint8  
 38  Retired              10938 non-null  uint8  
 39  Self-Employed        10938 non-null  uint8  
 40  Student              10938 non-null  uint8  
 41  Unemployed           10938 non-null  uint8  
 42  Education_not-given  10938 non-null  uint8  
 43  Graduate             10938 non-null  uint8  
 44  Post-Graduate        10938 non-null  uint8  
 45  Primary              10938 non-null  uint8  
 46  Secondary            10938 non-null  uint8  
dtypes: float64(10), int64(4), uint8(33)
memory usage: 1.6 MB
```

In [143…

```
X_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2735 entries, 5339 to 12030
Data columns (total 47 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   loannumber            2735 non-null   int64  
 1   loanamount            2735 non-null   float64
 2   totaldue              2735 non-null   float64
 3   termdays              2735 non-null   int64  
 4   perf_loannumber       2735 non-null   int64  
 5   perf_loanamount       2735 non-null   float64
 6   perf_totaldue         2735 non-null   float64
 7   perf_termdays         2735 non-null   int64  
 8   longitude_gps         2735 non-null   float64
 9   latitude_gps          2735 non-null   float64
 10  age                   2735 non-null   float64
 11  loanlife              2735 non-null   float64
 12  LateFirstPay          2735 non-null   float64
 13  EarlyFirstPay         2735 non-null   float64
 14  Current               2735 non-null   uint8  
 15  Other                 2735 non-null   uint8  
 16  Savings               2735 non-null   uint8  
 17  Access Bank           2735 non-null   uint8  
 18  Diamond Bank          2735 non-null   uint8  
 19  EcoBank               2735 non-null   uint8  
 20  FCMB                  2735 non-null   uint8  
 21  Fidelity Bank         2735 non-null   uint8  
 22  First Bank            2735 non-null   uint8  
 23  GT Bank               2735 non-null   uint8  
 24  Heritage Bank         2735 non-null   uint8  
 25  Keystone Bank         2735 non-null   uint8  
 26  Skye Bank             2735 non-null   uint8  
 27  Stanbic IBTC          2735 non-null   uint8  
 28  Standard Chartered    2735 non-null   uint8  
 29  Sterling Bank         2735 non-null   uint8  
 30  UBA                   2735 non-null   uint8  
 31  Union Bank            2735 non-null   uint8  
 32  Unity Bank            2735 non-null   uint8  
 33  Wema Bank             2735 non-null   uint8  
 34  Zenith Bank           2735 non-null   uint8  
 35  Contract              2735 non-null   uint8  
 36  Employment_not-given  2735 non-null   uint8  
 37  Permanent             2735 non-null   uint8  
 38  Retired               2735 non-null   uint8  
 39  Self-Employed         2735 non-null   uint8  
 40  Student               2735 non-null   uint8  
 41  Unemployed            2735 non-null   uint8  
 42  Education_not-given   2735 non-null   uint8  
 43  Graduate              2735 non-null   uint8  
 44  Post-Graduate         2735 non-null   uint8  
 45  Primary               2735 non-null   uint8  
 46  Secondary             2735 non-null   uint8  
dtypes: float64(10), int64(4), uint8(33)
memory usage: 408.6 KB
```

No missing values in the training and testing data

Scaling of the data

In [143…
```
cols = X_train.columns
```

In [143…
```
#Going to use a standard scaler
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In the next section, we are going to be building our model. From earlier in our notebook, we noticed that our data was imbalanced with the number of bad loans much smaller than the number of good loans. This doesn't give our models enough data to learn from about what a bad loan could look like.

We are going to deal with the imbalanced data but first, we want to see what would happen if we trained our data with the imbalanced dataset.

Logistic Regression

with imbalanced data

The first model we are building is a logistic regression model and it is dealing with the imbalanced data.

In [143…

```
#Building our logistic regression model
logreg = LogisticRegression(solver='liblinear', random_state=0)

logreg.fit(X_train, y_train)
```

Out[1433]:

```
▾ LogisticRegression
LogisticRegression(random_state=0, solver='liblinear')
```

In [143…

```
#Providing our testing data for predictions
y_pred_test = logreg.predict(X_test)
```

In [143…

```
#looking at the predictions given by the model
y_pred_test
```

Out[1435]:

```
array(['Good', 'Good', 'Good', ..., 'Good', 'Good', 'Good'], dtype=object)
```

In [143…

```
print('Model accuracy score(test): ', accuracy_score(y_test, y_pred_test))
```

```
Model accuracy score(test):  0.83327239488117
```

In [143…

```
y_pred_train = logreg.predict(X_train)

print('Model accuracy score(train): ', accuracy_score(y_train, y_pred_train))
```

```
Model accuracy score(train):  0.811848601206802
```

From the two cells above, we can see that the model's accuracy seems to be very high. This seems very strange considering we have imbalanced data

In [143…

```
y_test.value_counts()
```

Out[1438]:

```
Good    2275
Bad      460
Name: good_bad_flag, dtype: int64
```

In [143…

```
#confusion matrix for our model
cm =confusion_matrix(y_test, y_pred_test)
```

In [144…

```
cm
```

Out[1440]:

```
array([[  11,  449],
       [   7, 2268]])
```

In [144…

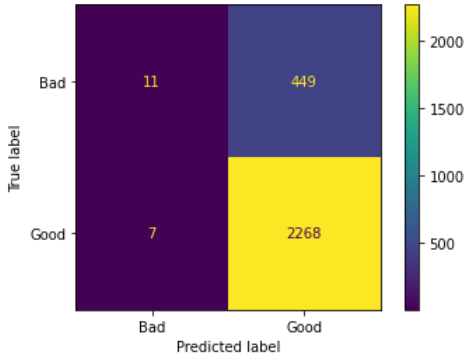
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=logreg.classes_)
```

In [144…

```
disp.plot()
```

Out[1442]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fd0a1a971c0>
```



From the confusion matrix above, we can see that the model is predicting good loans correctly: 2268 good loans were correctly precdicted with only 7 missclassified.

We can also see that it is only accurately predicted only 11 bad loans as bad. 449 bad loans were wrongly classified as good loans. This can be attributed to the fact that the model has a lot of data on good loans but not enough on bad loans.

In [144…

```
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

         Bad       0.61      0.02      0.05       460
        Good       0.83      1.00      0.91      2275

    accuracy                           0.83      2735
   macro avg       0.72      0.51      0.48      2735
weighted avg       0.80      0.83      0.76      2735
```

From the classification report above, we can still see the model doing very well on the good loans but very poorly for the bad loans We see we only have 460 instances of bad loans to train on. The macro average is low also show that data is imbalanced.

Dealing with imbalanced data

Now that we've seen the effect of imbalanced data, we can deal with this by creating new data from the existing data that we can then use to train our model.

In [144…

```
X = newdf.drop(['good_bad_flag'], axis=1)
y = newdf['good_bad_flag']
```

In [144…

```
#SMOTE is an oversampling technique that creates new data from existing
#for the minority class in this case, bad loans
#the sampling strategy chosen is all, no change with it being 'minority'
smote_algo = SMOTE(sampling_strategy='all',random_state=0)
smote_data_X, smote_data_Y = smote_algo.fit_resample(X,y)
smote_data_X = pd.DataFrame(data=smote_data_X, columns=X.columns)
smote_data_Y = pd.DataFrame(data=smote_data_Y, columns=['good_bad_flag'])
```

In [144…

```
smote_data = smote_data_X
smote_data['good_bad_flag'] = smote_data_Y['good_bad_flag']
```

In [150…

```
smote_data_Y['good_bad_flag'].value_counts()
```

Out[1504]:

```
Good    11146
Bad     11146
Name: good_bad_flag, dtype: int64
```

In [144…

```
smote_data.drop_duplicates(keep="first", inplace=True) #removing duplicate data if any
smote_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 22292 entries, 0 to 22291
Data columns (total 48 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   loannumber            22292 non-null  int64  
 1   loanamount            22292 non-null  float64
 2   totaldue              22292 non-null  float64
 3   termdays              22292 non-null  int64  
 4   perf_loannumber       22292 non-null  int64  
 5   perf_loanamount       22292 non-null  float64
 6   perf_totaldue         22292 non-null  float64
 7   perf_termdays         22292 non-null  int64  
 8   longitude_gps         22292 non-null  float64
 9   latitude_gps          22292 non-null  float64
 10  age                   22292 non-null  float64
 11  loanlife              22292 non-null  float64
 12  LateFirstPay          22292 non-null  float64
 13  EarlyFirstPay         22292 non-null  float64
 14  Current               22292 non-null  uint8  
 15  Other                 22292 non-null  uint8  
 16  Savings               22292 non-null  uint8  
 17  Access Bank           22292 non-null  uint8  
 18  Diamond Bank          22292 non-null  uint8  
 19  EcoBank               22292 non-null  uint8  
 20  FCMB                  22292 non-null  uint8  
 21  Fidelity Bank         22292 non-null  uint8  
 22  First Bank            22292 non-null  uint8  
 23  GT Bank               22292 non-null  uint8  
 24  Heritage Bank         22292 non-null  uint8  
 25  Keystone Bank         22292 non-null  uint8  
 26  Skye Bank             22292 non-null  uint8  
 27  Stanbic IBTC          22292 non-null  uint8  
 28  Standard Chartered    22292 non-null  uint8  
 29  Sterling Bank         22292 non-null  uint8  
 30  UBA                   22292 non-null  uint8  
 31  Union Bank            22292 non-null  uint8  
 32  Unity Bank            22292 non-null  uint8  
 33  Wema Bank             22292 non-null  uint8  
 34  Zenith Bank           22292 non-null  uint8  
 35  Contract              22292 non-null  uint8  
 36  Employment_not-given  22292 non-null  uint8  
 37  Permanent             22292 non-null  uint8  
 38  Retired               22292 non-null  uint8  
 39  Self-Employed         22292 non-null  uint8  
 40  Student               22292 non-null  uint8  
 41  Unemployed            22292 non-null  uint8  
 42  Education_not-given   22292 non-null  uint8  
 43  Graduate              22292 non-null  uint8  
 44  Post-Graduate         22292 non-null  uint8  
 45  Primary               22292 non-null  uint8  
 46  Secondary             22292 non-null  uint8  
 47  good_bad_flag         22292 non-null  object 
dtypes: float64(10), int64(4), object(1), uint8(33)
memory usage: 3.4+ MB
```

After applying SMOTE, we now have over 20,000 entries as compared to the 10,000 we had originally trained with for the first model instance. Now we can go ahead and train a new model using this more balanced data.

In [144…
```
X = smote_data.drop(['good_bad_flag'],axis=1)
y = smote_data['good_bad_flag']
```

In [144…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state= 2)
```

In [145…
```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [145…
```
X_train.shape, X_test.shape
```

Out[1451]:
```
((17833, 47), (4459, 47))
```

In [145…
```
logreg = LogisticRegression(solver='liblinear', random_state=3)
```

In [145…
```
logreg.fit(X_train, y_train)
```

Out[1453]:
```
▾ LogisticRegression
LogisticRegression(random_state=3, solver='liblinear')
```

In [145…
```
y_pred_test = logreg.predict(X_test)
y_pred_test
```

Out[1454]:
```
array(['Good', 'Bad', 'Good', ..., 'Bad', 'Bad', 'Good'], dtype=object)
```

In [145…
```
print('Model accuracy score(test): ', accuracy_score(y_test, y_pred_test))
```

```
Model accuracy score(test):  0.8178963893249608
```

In [145…
```
y_pred_train = logreg.predict(X_train)

print('Model accuracy score(train): ', accuracy_score(y_train, y_pred_train))
```

```
Model accuracy score(train):  0.8082207144058767
```

In [145…
```
y_test.value_counts()
```

Out[1457]:
```
Bad     2245
Good    2214
Name: good_bad_flag, dtype: int64
```

In [145…
```
null_accuracy = 2245/(len(y_test))
```

In [145…
```
null_accuracy
```

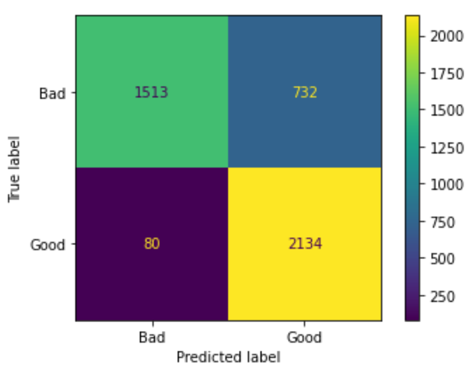
Out[1459]:
```
0.5034761157210137
```

In [146…
```
cm1 =confusion_matrix(y_test, y_pred_test)
```

In [146…
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm1, display_labels=logreg.classes_)
```

In [146…
```
disp.plot()
```

Out[1462]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fd0a196ab50>
```



From the matrix above, we can see that with this balance data set, the model was able to correctly predict 1513 bad loans. Although this is better than our initial model, the model still misclassified 732 bad loans as good.

The performance of the model at predicting good loans is still high.

In [146…
```
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

         Bad       0.95      0.67      0.79      2245
        Good       0.74      0.96      0.84      2214

    accuracy                           0.82      4459
   macro avg       0.85      0.82      0.81      4459
weighted avg       0.85      0.82      0.81      4459
```

Since our data is now balanced, we can see the macro average increased as well.

We can see the model is performing a little better, not yet best, at predicting bad loans. We can adjust the thresholds to help it perform better.

In [146…

```python
#getting the bad and good loan probabilities for each of our testing data
y_pred_prob = logreg.predict_proba(X_test)
y_pred_prob
```

Out[1464]:

```
array([[2.88240441e-01, 7.11759559e-01],
       [9.84949318e-01, 1.50506824e-02],
       [4.67154804e-01, 5.32845196e-01],
       ...,
       [9.99955936e-01, 4.40635405e-05],
       [6.59610105e-01, 3.40389895e-01],
       [1.16775187e-01, 8.83224813e-01]])
```

In [146…

```python
#storing the probabilities in a dataframe
y_pred_prob_df = pd.DataFrame(data=y_pred_prob, columns=['Probability of Good loan (0)',
                                                          'Probability of Bad loan (1)'])

y_pred_prob_df
```

Out[1465]:

	Probability of Good loan (0)	Probability of Bad loan (1)
0	0.288240	0.711760
1	0.984949	0.015051
2	0.467155	0.532845
3	0.377028	0.622972
4	0.369550	0.630450
...	...	...
4454	0.198785	0.801215
4455	0.996351	0.003649
4456	0.999956	0.000044
4457	0.659610	0.340390
4458	0.116775	0.883225

4459 rows × 2 columns

In [146…

```python
y_pred1 = logreg.predict_proba(X_test)[:,1]
y_pred0 = logreg.predict_proba(X_test)[:,0]
```

In [146…

```python
y_pred1.shape
```

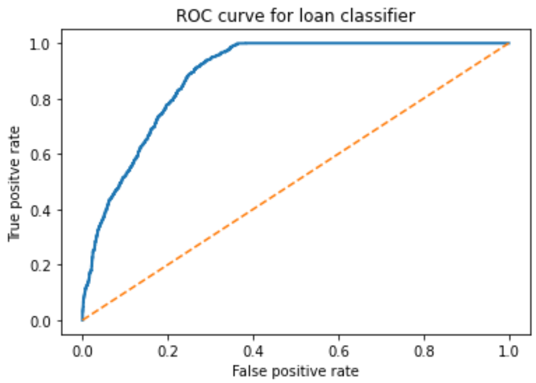
Out[1467]:

```
(4459,)
```

In [150…

```python
#plot ROC curve

from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_pred1, pos_label='Bad')
```

In [150...
```
plt.figure(figsize = (6,4))
plt.plot(fpr, tpr, linewidth=2)
plt.plot([0,1], [0,1], '--')
plt.title('ROC curve for loan classifier')
plt.xlabel('False positive rate')
plt.ylabel('True positve rate')
plt.show()
```



In [151...
```
#computing AUC

from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test, y_pred1)

print(ROC_AUC)
```

```
0.8848612695481076
```

In [151...
```
thresholds
```

Out[1511]:
```
array([1.99987749e+00, 9.99877486e-01, 9.53399127e-01, ...,
       1.70332434e-02, 1.64644241e-02, 2.96206025e-11])
```

In [149...
```
len(thresholds)
```

Out[1499]:
```
1094
```

In [148...
```
#y_pred1 = logreg.predict_proba(X_test)[:,1]
y_pred_prob= y_pred1.reshape(1,-1)
```

In [148...
```
y_pred_prob
```

Out[1486]:
```
array([[7.11759559e-01, 1.50506824e-02, 5.32845196e-01, ...,
        4.40635405e-05, 3.40389895e-01, 8.83224813e-01]])
```

In [148...
```
y_pred_prob.shape
```

Out[1487]:
```
(1, 4459)
```

Tuning the threshold to improve model performance. The threshold can changed multiple times to see if the performance of the logistic model's prediction will improve.

In [151...
```
#first testing of threshold
y_pred_class = binarize(y_pred_prob, threshold=1.99987749e+00)[0]
y_pred_class
```

Out[1512]:
```
array([0., 0., 0., ..., 0., 0., 0.])
```

In [151...
```
y_test
```

Out[1513]:
```
3092     Good
17257     Bad
10882    Good
9413     Good
11331    Good
         ...
3321     Good
15499     Bad
17287     Bad
473       Bad
7265     Good
Name: good_bad_flag, Length: 4459, dtype: object
```

In [151…
```
y_test1 = y_test.apply(lambda x: 0 if x == 'Good' else 1)
y_test1
```

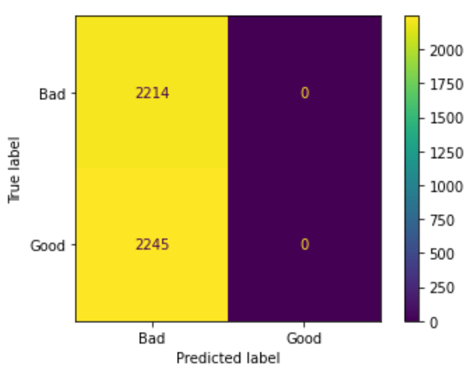
Out[1514]:
```
3092     0
17257    1
10882    0
9413     0
11331    0
        ..
3321     0
15499    1
17287    1
473      1
7265     0
Name: good_bad_flag, Length: 4459, dtype: int64
```

In [151…
```
cm = confusion_matrix(y_test1, y_pred_class)
```

In [151…
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=logreg.classes_)
```

In [151…
```
disp.plot()
```

Out[1518]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fd0a290d4f0>



In [149…
```
best_matrix=cm1
best_matrix
```

Out[1495]:
```
array([[1513,  732],
       [  80, 2134]])
```

In [149…
```
optimal_Threshold=0
```

In [149…
```
#this function is looping through the given thresholds and checking to see
# if there exists a threshold that can give us more true positives and true negatives
# as well as less false negatives
for thresh in thresholds:
    y_pred_class = binarize(y_pred_prob, threshold=thresh)[0]
    cm = confusion_matrix(y_test1, y_pred_class)
    if((cm[0,0]>best_matrix[0,0]) and cm[0,1]<best_matrix[0,1] and cm[1,1]>=best_matrix[1,1]):
        optimal_Threshold = thresh
        best_matrix=cm
```

In [149…
```
optimal_Threshold, best_matrix
```
Out[1498]:
```
(0,
 array([[1513,  732],
        [  80, 2134]]))
```

From the results above, we can see that there was no optimal threshold returned from the list of thresholds. So the first iteration of our regression model seems to be the best that logistic regression can perform perform to on this data.

K-nearest neighbours

In [732…
```
#we're going to build three models testing out a different number of nearest neighbours to
#see which number is best performing

#model where n_neighbors = 11
knnr = KNeighborsClassifier(n_neighbors = 11, metric='minkowski',p=2)
```

In [733…
```
knnr.fit(X_train, y_train)
```
Out[733]:
```
▾        KNeighborsClassifier
KNeighborsClassifier(n_neighbors=11)
```

In [734…
```
y_pred = knnr.predict(X_test)
```

In [735…
```
print(classification_report(y_test, y_pred))
```
```
              precision    recall  f1-score   support

         Bad       0.89      0.86      0.88      2222
        Good       0.87      0.90      0.88      2237

    accuracy                           0.88      4459
   macro avg       0.88      0.88      0.88      4459
weighted avg       0.88      0.88      0.88      4459
```

For n_neighbours=11, we can see the model is performing equally well predicting the loan classes. Can we have better scores though? In the next cell we have accuracies and classification reports for n_neighbours=15, 21, 5,3

n_neighbours = 15

	precision	recall	f1-score	support
Bad	0.88	0.86	0.87	2245
Good	0.87	0.88	0.87	2214
accuracy			0.87	4459
macro avg	0.87	0.87	0.87	4459
weighted avg	0.87	0.87	0.87	4459

n_neighbours = 21

	precision	recall	f1-score	support
Bad	0.89	0.83	0.86	2245
Good	0.84	0.89	0.87	2214
accuracy			0.86	4459
macro avg	0.86	0.86	0.86	4459
weighted avg	0.86	0.86	0.86	4459

n_neighbours = 5

	precision	recall	f1-score	support
Bad	0.90	0.92	0.91	2245
Good	0.92	0.90	0.91	2214
accuracy			0.91	4459
macro avg	0.91	0.91	0.91	4459
weighted avg	0.91	0.91	0.91	4459

n_neighbours = 3

	precision	recall	f1-score	support
Bad	0.91	0.94	0.92	2245
Good	0.93	0.91	0.92	2214
accuracy			0.92	4459
macro avg	0.92	0.92	0.92	4459
weighted avg	0.92	0.92	0.92	4459

From the reports and accuracies above, we can see that n_neighbours=3 yields the best results for our loan classifier for both identifiying the good and bad loans.

Let us see if n_neighbours = 1 will yield even better results

n_neighbours = 1

	precision	recall	f1-score	support
Bad	0.92	0.94	0.93	2245
Good	0.94	0.92	0.93	2214
accuracy			0.93	4459
macro avg	0.93	0.93	0.93	4459
weighted avg	0.93	0.93	0.93	4459

n_neighbours =1 gives even better results and so we shall take n_neighbours=1 for the knn model.

Support vector machine

In [710...
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state= 5)
```

In [711...
```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [712...
```
classifier = SVC(kernel='linear', random_state=0)
```

In [713...
```
classifier.fit(X_train, y_train)
```

Out[713]:
```
▾ SVC
SVC(kernel='linear', random_state=0)
```

In [714...
```
y_pred= classifier.predict(X_test)
y_pred
```

Out[714]:
```
array(['Good', 'Bad', 'Good', ..., 'Good', 'Good', 'Good'], dtype=object)
```

In [715…
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         Bad       1.00      0.63      0.77      2222
        Good       0.73      1.00      0.84      2237

    accuracy                           0.81      4459
   macro avg       0.86      0.81      0.81      4459
weighted avg       0.86      0.81      0.81      4459
```

Model selection

model	f1-score(bad loans)	f1-score(good loans)
Logistic regression	0.79	0.84
KNN	0.93	0.93
SVM	0.77	0.84

Conlusion: Of the three models, KNN with n_neighbours=1 seems to be performing the best and that is what we would provide as a loan classifier to Super Digital