Credit card project

Project - To predict their clients' repayment abilities—

Table of content:-

Import Necessary Libraries in python notebook

- For analysis using python we have to use pandas library which efficient in data frame
- Matplotlib library is used to plot charts for the data set.
- Numpy arrays are faster and more compact than python lists
- Import pandas as pd
- Import matplotlib.pyplot as plt
- Import as numpy as np

➢ Import the csv data set in notebook

- We have been given 8 csv files of credit card details
- To import csv files we hve to use command as
- application_test = pd.read_csv("application_test.csv")
- application_train = pd.read_csv("application_train.csv")
- bureau= pd.read_csv("bureau.csv")
- bureau_balance = pd.read_csv("bureau_balance.csv")
- installments_payments = pd.read_csv("installments_payments.csv")

- POS_CASH_balance= pd.read_csv("POS_CASH_balance.csv")
- previous_application = pd.read_csv("previous_application.csv")
- credit_card_balance = pd.read_csv("credit_card_balance.csv")

Check Data

- to check the data types of a column in a data frame we use dtypes.
- application_train.dtypes

```
In [14]: #analyse datset
         application_train.dtypes

Out[14]: SK_ID_CURR                     int64
         TARGET                         int64
         NAME_CONTRACT_TYPE            object
         CODE_GENDER                   object
         FLAG_OWN_CAR                  object
                                        ...
         AMT_REQ_CREDIT_BUREAU_DAY    float64
         AMT_REQ_CREDIT_BUREAU_WEEK   float64
         AMT_REQ_CREDIT_BUREAU_MON    float64
         AMT_REQ_CREDIT_BUREAU_QRT    float64
         AMT_REQ_CREDIT_BUREAU_YEAR   float64
         Length: 122, dtype: object
```

- POS_CASH_balance.dtypes

```
SK_ID_PREV                 int64
SK_ID_CURR                 int64
MONTHS_BALANCE             int64
CNT_INSTALMENT           float64
CNT_INSTALMENT_FUTURE    float64
NAME_CONTRACT_STATUS      object
SK_DPD                     int64
SK_DPD_DEF                 int64
dtype: object
```

bureau_balance.dtypes

In [16]:
```
#analyse dataset
bureau_balance.dtypes
```

Out[16]:
```
SK_ID_BUREAU       int64
MONTHS_BALANCE     int64
STATUS            object
dtype: object
```

previous_application.dtypes

In [17]:
```
#analyse dataset
previous_application.dtypes
```

Out[17]:
```
SK_ID_PREV                       int64
SK_ID_CURR                       int64
NAME_CONTRACT_TYPE              object
AMT_ANNUITY                    float64
AMT_APPLICATION                float64
AMT_CREDIT                     float64
AMT_DOWN_PAYMENT               float64
AMT_GOODS_PRICE                float64
WEEKDAY_APPR_PROCESS_START      object
HOUR_APPR_PROCESS_START          int64
FLAG_LAST_APPL_PER_CONTRACT     object
NFLAG_LAST_APPL_IN_DAY           int64
RATE_DOWN_PAYMENT              float64
RATE_INTEREST_PRIMARY          float64
RATE_INTEREST_PRIVILEGED       float64
NAME_CASH_LOAN_PURPOSE          object
NAME_CONTRACT_STATUS            object
DAYS_DECISION                    int64
NAME_PAYMENT_TYPE               object
CODE_REJECT_REASON              object
NAME_TYPE_SUITE                 object
```

installments_payments.dtypes

```
In [18]: installments_payments.dtypes
         #analyse dataset
```

```
Out[18]: SK_ID_PREV                  int64
         SK_ID_CURR                  int64
         NUM_INSTALMENT_VERSION    float64
         NUM_INSTALMENT_NUMBER       int64
         DAYS_INSTALMENT           float64
         DAYS_ENTRY_PAYMENT        float64
         AMT_INSTALMENT            float64
         AMT_PAYMENT               float64
         dtype: object
```

credit_card_balance.dtypes

```
In [19]: credit_card_balance.dtypes
```

```
Out[19]: SK_ID_PREV                      int64
         SK_ID_CURR                      int64
         MONTHS_BALANCE                  int64
         AMT_BALANCE                   float64
         AMT_CREDIT_LIMIT_ACTUAL         int64
         AMT_DRAWINGS_ATM_CURRENT      float64
         AMT_DRAWINGS_CURRENT          float64
         AMT_DRAWINGS_OTHER_CURRENT    float64
         AMT_DRAWINGS_POS_CURRENT      float64
         AMT_INST_MIN_REGULARITY       float64
         AMT_PAYMENT_CURRENT           float64
         AMT_PAYMENT_TOTAL_CURRENT     float64
         AMT_RECEIVABLE_PRINCIPAL      float64
         AMT_RECIVABLE                 float64
         AMT_TOTAL_RECEIVABLE          float64
         CNT_DRAWINGS_ATM_CURRENT      float64
         CNT_DRAWINGS_CURRENT            int64
         CNT_DRAWINGS_OTHER_CURRENT    float64
         CNT_DRAWINGS_POS_CURRENT      float64
         CNT_INSTALMENT_MATURE_CUM     float64
         NAME_CONTRACT_STATUS           object
```

➢ Check for missing data

- **isnull()**. values. any() will work for a DataFrame object to indicate if any value is missing , in some cases it may be useful to also count the number of missing values across the entire DataFrame.
- . count = application_train.isnull().mean().sort_values(ascending=False).head(50)
- count

In [81]:
```
# checking missing data

count = application_train.isnull().mean().sort_values(ascending=False).head(50)
count
```

Out[81]:
```
COMMONAREA_MEDI                 0.698723
COMMONAREA_AVG                  0.698723
COMMONAREA_MODE                 0.698723
NONLIVINGAPARTMENTS_MODE        0.694330
NONLIVINGAPARTMENTS_AVG         0.694330
NONLIVINGAPARTMENTS_MEDI        0.694330
FONDKAPREMONT_MODE              0.683862
LIVINGAPARTMENTS_MODE           0.683550
LIVINGAPARTMENTS_AVG            0.683550
LIVINGAPARTMENTS_MEDI           0.683550
FLOORSMIN_AVG                   0.678486
FLOORSMIN_MODE                  0.678486
FLOORSMIN_MEDI                  0.678486
YEARS_BUILD_MEDI                0.664978
YEARS_BUILD_MODE                0.664978
YEARS_BUILD_AVG                 0.664978
OWN_CAR_AGE                     0.659908
LANDAREA_MEDI                   0.593767
LANDAREA_MODE                   0.593767
LANDAREA_AVG                    0.593767
BASEMENTAREA_MEDI               0.585160
```

checking missing data

POS_CASH_balance.isna().sum()

count = POS_CASH_balance.isnull().sum().sort_values(ascending=False)

percentage = ((POS_CASH_balance.isnull().sum()/len(POS_CASH_balance)*100)).sort_values(ascending=False)

missing_POS_CASH_balance = pd.concat([count, percentage], axis=1, keys=['Count','Percentage'])

print('Count and percentage of missing values for top 20 columns:')

missing_POS_CASH_balance.head(20)

In [24]:
```
# checking missing data
POS_CASH_balance.isna().sum()
count = POS_CASH_balance.isnull().sum().sort_values(ascending=False)
percentage = ((POS_CASH_balance.isnull().sum()/len(POS_CASH_balance)*100)).sort_values(ascending=False)
missing_POS_CASH_balance = pd.concat([count, percentage], axis=1, keys=['Count','Percentage'])
print('Count and percentage of missing values for top 20 columns:')
missing_POS_CASH_balance.head(20)
```

```
Count and percentage of missing values for top 20 columns:
```

Out[24]:

	Count	Percentage
CNT_INSTALMENT_FUTURE	26087	0.260835
CNT_INSTALMENT	26071	0.260675
SK_ID_PREV	0	0.000000
SK_ID_CURR	0	0.000000
MONTHS_BALANCE	0	0.000000
NAME_CONTRACT_STATUS	0	0.000000
SK_DPD	0	0.000000
SK_DPD_DEF	0	0.000000

checking missing data

checking missing data   prev_app

count =bureau_balance.isnull().sum().sort_values(ascending=False)

percentage = ((bureau_balance.isnull().sum()/len(bureau_balance)*100)).sort_values(ascending=False)

missing_bureau_balance = pd.concat([count, percentage], axis=1, keys=['Count','Percentage'])

print('Count and percentage of missing values for top 20 columns:')

missing_bureau_balance.head(20)

In [26]:
```
# checking missing data

# checking missing data    prev_app
count =bureau_balance.isnull().sum().sort_values(ascending=False)
percentage = ((bureau_balance.isnull().sum()/len(bureau_balance)*100)).sort_values(ascending=False)
missing_bureau_balance = pd.concat([count, percentage], axis=1, keys=['Count','Percentage'])
print('Count and percentage of missing values for top 20 columns:')
missing_bureau_balance.head(20)
```

Count and percentage of missing values for top 20 columns:

Out[26]:

	Count	Percentage
SK_ID_BUREAU	0	0.0
MONTHS_BALANCE	0	0.0
STATUS	0	0.0

checking missing data

previous_application.isna().sum()

count =previous_application.isnull().sum().sort_values(ascending=False)

percentage =((previous_application.isnull().sum()/len(previous_application)*100)).sort_values(ascending=False)

missing_previous_application = pd.concat([count, percentage], axis=1, keys=['Count','Percentage'])

print('Count and percentage of missing values for top 20 columns:')

missing_previous_application.head(20)

In [29]:
```
# checking missing data
previous_application.isna().sum()
count =previous_application.isnull().sum().sort_values(ascending=False)
percentage =((previous_application.isnull().sum()/len(previous_application)*100)).sort_values(ascending=False)
missing_previous_application = pd.concat([count, percentage], axis=1, keys=['Count','Percentage'])
print('Count and percentage of missing values for top 20 columns:')
missing_previous_application.head(20)
```

Count and percentage of missing values for top 20 columns:

Out[29]:

	Count	Percentage
RATE_INTEREST_PRIVILEGED	1664263	99.643698
RATE_INTEREST_PRIMARY	1664263	99.643698
AMT_DOWN_PAYMENT	895844	53.636480
RATE_DOWN_PAYMENT	895844	53.636480
NAME_TYPE_SUITE	820405	49.119754
NFLAG_INSURED_ON_APPROVAL	673065	40.298129
DAYS_TERMINATION	673065	40.298129
DAYS_LAST_DUE	673065	40.298129
DAYS_LAST_DUE_1ST_VERSION	673065	40.298129
DAYS_FIRST_DUE	673065	40.298129

checking missing data

installments_payments.isna().sum()

count =installments_payments.isnull().sum().sort_values(ascending=False)

percentage =((installments_payments.isnull().sum()/len(installments_payments)*100)).sort_values(ascending=False)

missing_installments_payments = pd.concat([count, percentage], axis=1, keys=['Count','Percentage'])

print('Count and percentage of missing values for top 20 columns:')

missing_installments_payments.head(20)

```
In [30]: # checking missing data
         installments_payments.isna().sum()
         count =installments_payments.isnull().sum().sort_values(ascending=False)
         percentage =((installments_payments.isnull().sum()/len(installments_payments)*100)).sort_values(ascending=False)
         missing_installments_payments = pd.concat([count, percentage], axis=1, keys=['Count','Percentage'])
         print('Count and percentage of missing values for top 20 columns:')
         missing_installments_payments.head(20)

         Count and percentage of missing values for top 20 columns:
Out[30]:
```

	Count	Percentage
DAYS_ENTRY_PAYMENT	2905	0.021352
AMT_PAYMENT	2905	0.021352
SK_ID_PREV	0	0.000000
SK_ID_CURR	0	0.000000
NUM_INSTALMENT_VERSION	0	0.000000
NUM_INSTALMENT_NUMBER	0	0.000000
DAYS_INSTALMENT	0	0.000000
AMT_INSTALMENT	0	0.000000

checking missing data

count =credit_card_balance.isnull().sum().sort_values(ascending=False)

percentage =((credit_card_balance.isnull().sum()/len(credit_card_balance)*100)).sort_values(ascending=False)

missing_credit_card_balance = pd.concat([count, percentage], axis=1, keys=['Count','Percentage'])

print('Count and percentage of missing values for top 20 columns:')

missing_credit_card_balance.head(20)

*checking missing data in credit_card_balance *

```
In [32]: # checking missing data

         count =credit_card_balance.isnull().sum().sort_values(ascending=False)
         percentage =((credit_card_balance.isnull().sum()/len(credit_card_balance)*100)).sort_values(ascending=False)
         missing_credit_card_balance = pd.concat([count, percentage], axis=1, keys=['Count','Percentage'])
         print('Count and percentage of missing values for top 20 columns:')
         missing_credit_card_balance.head(20)

         Count and percentage of missing values for top 20 columns:
Out[32]:
```

	Count	Percentage
AMT_PAYMENT_CURRENT	767988	19.998063
AMT_DRAWINGS_ATM_CURRENT	749816	19.524872
CNT_DRAWINGS_POS_CURRENT	749816	19.524872
AMT_DRAWINGS_OTHER_CURRENT	749816	19.524872
AMT_DRAWINGS_POS_CURRENT	749816	19.524872
CNT_DRAWINGS_OTHER_CURRENT	749816	19.524872
CNT_DRAWINGS_ATM_CURRENT	749816	19.524872
CNT_INSTALMENT_MATURE_CUM	305236	7.948208
AMT_INST_MIN_REGULARITY	305236	7.948208

count =bureau.isnull().sum().sort_values(ascending=False)

percentage =((bureau.isnull().sum()/len(bureau)*100)).sort_values(ascending=False)

missing_bureau = pd.concat([count, percentage], axis=1, keys=['Count','Percentage'])

print('Count and percentage of missing values for top 20 columns:')

missing_bureau.head(20)

*checking missing data in bureau *

```
[33]: count =bureau.isnull().sum().sort_values(ascending=False)
percentage =((bureau.isnull().sum()/len(bureau)*100)).sort_values(ascending=False)
missing_bureau = pd.concat([count, percentage], axis=1, keys=['Count','Percentage']
print('Count and percentage of missing values for top 20 columns:')
missing_bureau.head(20)
```

```
Count and percentage of missing values for top 20 columns:
```

t[33]:

	Count	Percentage
AMT_ANNUITY	1226791	71.473490
AMT_CREDIT_MAX_OVERDUE	1124488	65.513264
DAYS_ENDDATE_FACT	633653	36.916958
AMT_CREDIT_SUM_LIMIT	591780	34.477415
AMT_CREDIT_SUM_DEBT	257669	15.011932
DAYS_CREDIT_ENDDATE	105553	6.149573
AMT_CREDIT_SUM	13	0.000757
CREDIT_ACTIVE	0	0.000000

➢ Data exploration

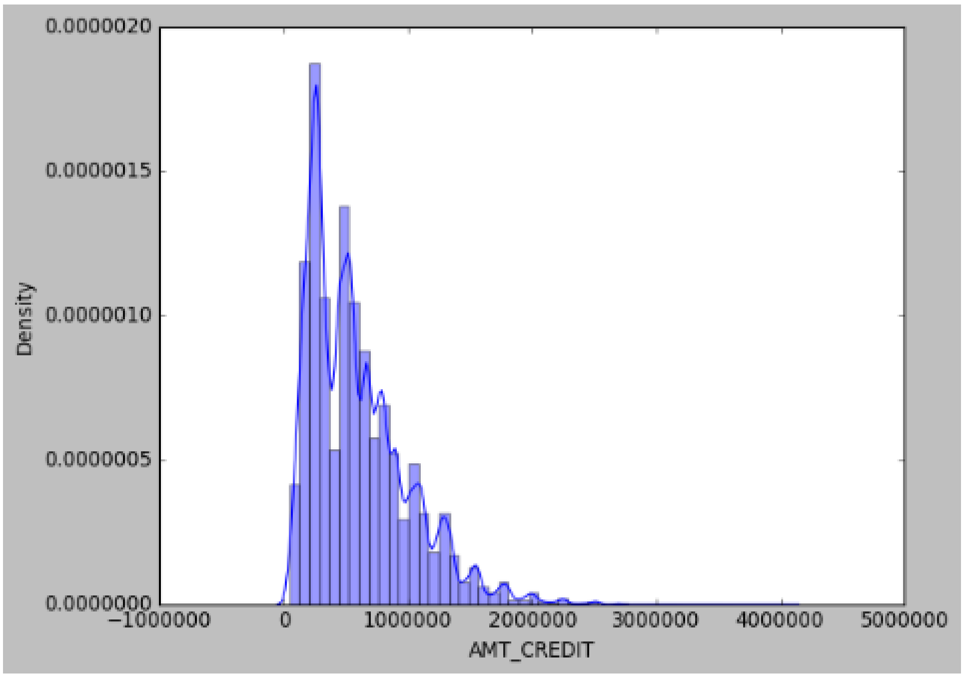
- Data exploration is a key aspect of data analysis and model building.
- Data exploration techniques include both manual analysis and automated data exploration software solutions that visually explore and identify relationships between different data variables.

Distribution of AMT_CREDIT

plt.style.use("classic")

sns.distplot(application_train["AMT_CREDIT"],bins = 50)

Out[28]: <AxesSubplot:xlabel='AMT_CREDIT', ylabel='Density'>



- Who accompanied client when applying for the application
  #code
  previous_application["NAME_TYPE_SUITE"].value_counts()

```
In [80]: #code
         previous_application["NAME_TYPE_SUITE"].value_counts()

Out[80]: Unaccompanied      508970
         Family             213263
         Spouse, partner     67069
         Children            31566
         Other_B             17624
         Other_A              9077
         Group of people      2240
         Name: NAME_TYPE_SUITE, dtype: int64
```

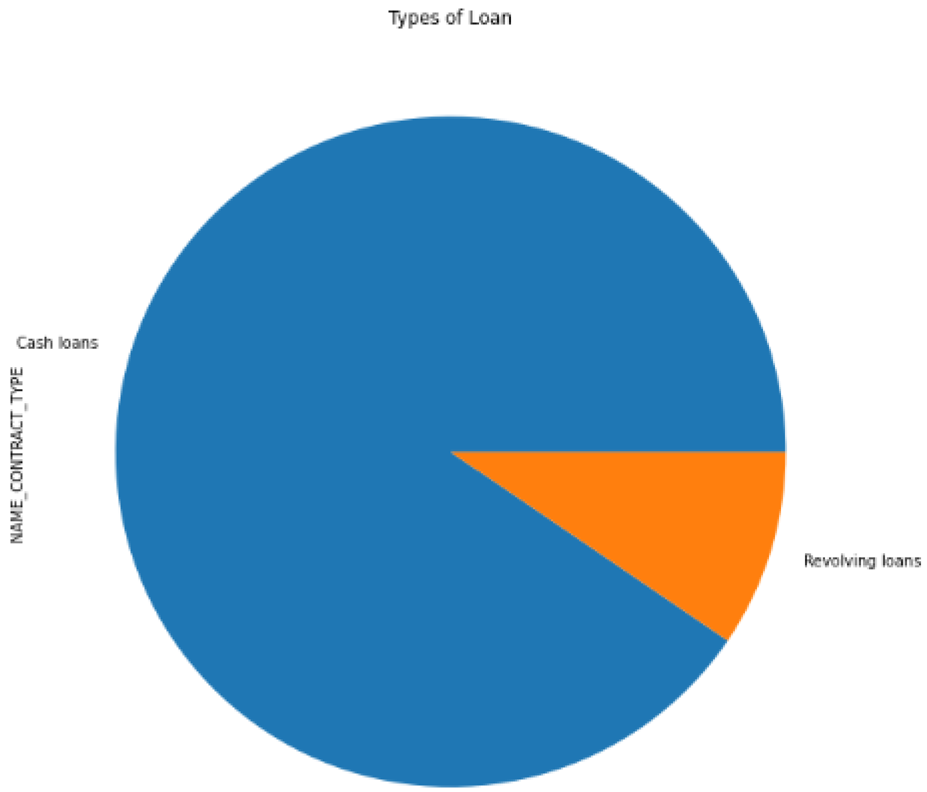
Types of loan

- Rovolving loans : Arrangement which allows for the loan amount to be withdrawn, repaid, and redrawn again in any manner and any number of times, until the arrangement expires. Credit card loans and overdrafts are revolving loans. Also called evergreen loan

  loan_type.plot.pie(title='Types of Loan',figsize=(10,12))

```
[84]: loan_type.plot.pie(title='Types of Loan',figsize=(10,12))
```

```
[84]: <AxesSubplot:title={'center':'Types of Loan'}, ylabel='NAME_CONTRACT_TYPE'>
```



Exploartion of previous application data

Contract product type of previous application

- We use value_counts() function.
  it is used to get a Series containing counts of unique values.
- With normalize set to True , returns the relative frequency by dividing all values by the sum of values
- We use unique () to get unique values of Series object. Uniques are returned in order of appearance.

```
In [45]: #code
previous_application['NAME_CONTRACT_TYPE'].value_counts(normalize=True) * 100

Out[45]: Cash loans         44.757917
Consumer loans     43.656142
Revolving loans    11.565225
XNA                 0.020716
Name: NAME_CONTRACT_TYPE, dtype: float64
```

- Contract product type of previous application :
  - Cash loans - 44.8 %
  - Consumer loans - 43.7 %
  - Rovolving loan - 11.6 %
  - XNA - 0.0207 %

On which day highest number of clients applied in prevoies application

previous_application['WEEKDAY_APPR_PROCESS_START'].value_counts(normalize=True) * 100

```
In [46]: 
previous_application['WEEKDAY_APPR_PROCESS_START'].value_counts(normalize=True) * 100

Out[46]: TUESDAY      15.274570
WEDNESDAY    15.268103
MONDAY       15.181109
FRIDAY       15.090761
THURSDAY     14.914197
SATURDAY     14.407196
SUNDAY        9.864065
Name: WEEKDAY_APPR_PROCESS_START, dtype: float64
```

- What a coincedence, Approximately 15 % clients applied in each 5 days a week i.e, Tuesday, Wednesday, Monday, Friday and Thrusday.

Purpose of cash loan in previous application

previous_application['NAME_SELLER_INDUSTRY'].value_counts(normalize=True) * 100

```
In [47]: previous_application['NAME_SELLER_INDUSTRY'].value_counts(normalize=True) * 100

Out[47]: XNA                     51.234153
Consumer electronics    23.845148
Connectivity            16.526565
Furniture                3.463568
Construction             1.783065
Clothing                 1.433888
Industry                 1.149194
Auto technology          0.298764
Jewelry                  0.162195
MLM partners             0.072745
Tourism                  0.030715
Name: NAME_SELLER_INDUSTRY, dtype: float64
```

- Main purpose of the cash loan was :
  - XAP - 55 %
  - XNA - 41 %

Contract was approved or not in previous application

df=previous_application['NAME_CASH_LOAN_PURPOSE'].value_counts(normalize=True)*100

```
In [50]: df=previous_application['NAME_CASH_LOAN_PURPOSE'].value_counts(normalize=True)*100
```

```
In [51]: df.head(4)

Out[51]: XAP        55.242083
         XNA        40.588691
         Repairs     1.422872
         Other       0.934491
         Name: NAME_CASH_LOAN_PURPOSE, dtype: float64
```

- Contract was approved or not in previous application :
  - Approved : 62.1 % times
  - Cancelled : 18.9 % times
  - Refused : 17.4 % times
  - Unused offer : 1.58 % times

Payment method that client choose to pay for the previous application

previous_application['NAME_PORTFOLIO'].value_counts(normalize=True)*100

```
In [52]: previous_application['NAME_PORTFOLIO'].value_counts(normalize=True)*100

Out[52]: POS       41.372603
         Cash      27.634962
         XNA       22.286366
         Cards      8.680624
         Cars       0.025446
         Name: NAME_PORTFOLIO, dtype: float64
```

- As we can most of the payment(61.9 %) has done thorugh cash only.

Why was the previous application rejected ?

previous_application['CODE_REJECT_REASON'].unique()

```
In [53]: previous_application['CODE_REJECT_REASON'].unique()

Out[53]: array(['XAP', 'HC', 'LIMIT', 'CLIENT', 'SCOFR', 'SCO', 'XNA', 'VERIF',
                'SYSTEM'], dtype=object)
```

Who accompanied client when applying for the previous application

previous_application['NAME_TYPE_SUITE'].value_counts(normalize=True)*100

```
In [55]: previous_application['NAME_TYPE_SUITE'].value_counts(normalize=True)*100

Out[55]: Unaccompanied      59.892282
         Family             25.095404
         Spouse, partner     7.892244
         Children            3.714482
         Other_B             2.073878
         Other_A             1.068122
         Group of people     0.263589
         Name: NAME_TYPE_SUITE, dtype: float64
```

- **Who accompanied client when applying for the previous application :**
  - Unaccompanied : Approx. 60 % times
  - Family : Approx. 25 % times
  - Spouse, Partner : Approx. 8 %
  - Childrens : Approx. 4 %

Was the client old or new client when applying for the previous application

previous_application['NAME_CLIENT_TYPE'].value_counts(normalize=True)*100

```
In [56]: previous_application['NAME_CLIENT_TYPE'].value_counts(normalize=True)*100

Out[56]: Repeater      73.718757
         New           18.043376
         Refreshed      8.121654
         XNA            0.116213
         Name: NAME_CLIENT_TYPE, dtype: float64
```

- Approximately 74 % was repeater clients who applied for previous application.

What kind of goods did the client apply for in the previous application

previous_application['NAME_GOODS_CATEGORY'].value_counts(normalize=True) * 100

```
In [64]: previous_application['NAME_GOODS_CATEGORY'].value_counts(normalize=True) * 100

Out[64]: XNA                         56.927376
         Mobile                      13.453845
         Consumer Electronics         7.279067
         Computers                    6.332662
         Audio/Video                  5.953788
         Furniture                    3.212522
         Photo / Cinema Equipment     1.498072
         Construction Materials       1.496515
         Clothing and Accessories     1.410238
         Auto Accessories             0.441919
         Jewelry                      0.376598
         Homewares                    0.300740
         Medical Supplies             0.230090
         Vehicles                     0.201771
         Sport and Leisure            0.178480
         Gardening                    0.159740
         Other                        0.152915
         Office Appliances            0.139683
         Tourism                      0.099329
         Medicine                     0.092802
         Direct Sales                 0.026703
         Fitness                      0.012513
         Additional Service           0.007664
         Education                    0.006406
         Weapon                       0.004610
         Insurance                    0.003832
         Animals                      0.000060
         House Construction           0.000060
         Name: NAME_GOODS_CATEGORY, dtype: float64
```

Was the previous application for CASH, POS, CAR, …

previous_application['NAME_PORTFOLIO'].value_counts(normalize=True) * 100

```
In [65]: previous_application['NAME_PORTFOLIO'].value_counts(normalize=True) * 100

Out[65]: POS      41.372603
         Cash     27.634962
         XNA      22.286366
         Cards     8.680624
         Cars      0.025446
         Name: NAME_PORTFOLIO, dtype: float64
```

Was the previous application x-sell or walk-in ?

previous_application['NAME_PRODUCT_TYPE'].value_counts(normalize=True) * 100

```
In [66]: previous_application['NAME_PRODUCT_TYPE'].value_counts(normalize=True) * 100

Out[66]: XNA        63.684414
         x-sell     27.319074
         walk-in     8.996512
         Name: NAME_PRODUCT_TYPE, dtype: float64
```

Top channels through which they acquired the client on the previous application

df2=previous_application['CHANNEL_TYPE'].value_counts(normalize=True) * 100

df2.head(4)

```
In [67]: df2=previous_application['CHANNEL_TYPE'].value_counts(normalize=True) * 100
         df2.head(4)

Out[67]: Credit and cash offices    43.106332
         Country-wide               29.618360
         Stone                      12.697954
         Regional / Local            6.497850
         Name: CHANNEL_TYPE, dtype: float64
```

- **Top channels through which they acquired the client on the previous application :**
  - Crediddit and cash offices : 43 % times
  - Country_wide : 30 % times
  - Stone : 13 % times

Top industry of the seller

df3=previous_application['NAME_SELLER_INDUSTRY'].value_counts(normalize=True) * 100

df3.head(2)

```
In [69]: df3=previous_application['NAME_SELLER_INDUSTRY'].value_counts(normalize=True) * 100
         df3.head(2)

Out[69]: XNA                     51.234153
         Consumer electronics    23.845148
         Name: NAME_SELLER_INDUSTRY, dtype: float64
```

Grouped interest rate into small medium and high of the previous application

previous_application['NAME_YIELD_GROUP'].value_counts(normalize=True) * 100

```
In [70]: previous_application['NAME_YIELD_GROUP'].value_counts(normalize=True) * 100

Out[70]: XNA           30.966990
         middle        23.082791
         high          21.154834
         low_normal    19.284655
         low_action     5.510731
         Name: NAME_YIELD_GROUP, dtype: float64
```

Top Detailed product combination of the previous application

previous_application['PRODUCT_COMBINATION'].value_counts(normalize=True) * 100

```
In [71]: previous_application['PRODUCT_COMBINATION'].value_counts(normalize=True) * 100

Out[71]: Cash                              17.126503
         POS household with interest       15.786996
         POS mobile with interest          13.214817
         Cash X-Sell: middle                8.616430
         Cash X-Sell: low                   7.799898
         Card Street                        6.741970
         POS industry with interest         5.918612
         POS household without interest     4.964943
         Card X-Sell                        4.825651
         Cash Street: high                  3.571480
         Cash X-Sell: high                  3.551239
         Cash Street: middle                2.075493
         Cash Street: low                   2.026148
         POS mobile without interest        1.442150
         POS other with interest            1.429993
         POS industry without interest      0.754670
         POS others without interest        0.153006
         Name: PRODUCT_COMBINATION, dtype: float64
```

Did the client requested insurance during the previous application

previous_application['NFLAG_INSURED_ON_APPROVAL'].value_counts(normalize=True) * 100

```
In [72]: previous_application['NFLAG_INSURED_ON_APPROVAL'].value_counts(normalize=True) * 100

Out[72]: 0.0    66.742984
         1.0    33.257016
         Name: NFLAG_INSURED_ON_APPROVAL, dtype: float64
```