Project Title: Home Credit Default Risk (HCDR)

The course project is based on the Home Credit Default Risk (HCDR) Kaggle Competition. The goal of this project is to predict whether or not a client will repay a loan. In order to make sure that people who struggle to get loans due to insufficient or non-existent credit histories have a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.

Some of the challenges

1. Dataset size
   - (688 meg compressed) with millions of rows of data
   - 2.71 Gig of data uncompressed

- Dealing with missing data
- Imbalanced datasets
- Summarizing transaction data

Back ground Home Credit Group

Many people struggle to get loans due to insufficient or non-existent credit histories. And, unfortunately, this population is often taken advantage of by untrustworthy lenders.

Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.

While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kagglers to help them unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful.

Project Title: Home Credit Default Risk (HCDR)

Exploratory Data Analysis

Dataset Size

In [6]:
```
for ds_name in datasets.keys():
    print(f'dataset {ds_name:24}: [ {datasets[ds_name].shape[0]:10,}, {datasets[ds_name].shape[1]}]')
```

```
dataset application_train       : [    307,511, 122]
dataset application_test        : [     48,744, 121]
dataset bureau                  : [  1,716,428, 17]
dataset bureau_balance          : [ 27,299,925, 3]
dataset credit_card_balance     : [  3,840,312, 23]
dataset installments_payments   : [ 13,605,401, 8]
dataset previous_application    : [  1,670,214, 37]
dataset POS_CASH_balance        : [  3,829,580, 8]
```

Function to plot the missing values

In [7]:
```
def plot_missing_data(df, x, y):
    g = sns.displot(
```

```
        data=datasets[df].isna().melt(value_name="missing"),
        y="variable",
        hue="missing",
        multiple="fill",
        aspect=1.25
    )
    g.fig.set_figwidth(x)
    g.fig.set_figheight(y)
```

Summary of Application train

In [15]:
```
datasets["application_train"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
```

In [16]:
```
datasets["application_train"].columns
```

Out[16]:
```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY',
       ...
       'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20',
       'FLAG_DOCUMENT_21', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR'],
      dtype='object', length=122)
```

In [17]:
```
datasets["application_train"].dtypes
```

Out[17]:
```
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
```

In [18]:
```
datasets["application_train"].describe() #numerical only features
```

Out[18]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_PO
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.000000	3.072330e+05	
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.573909	5.383962e+05	
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.737315	3.694465e+05	
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.500000	4.050000e+04	
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.000000	2.385000e+05	
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.000000	4.500000e+05	
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05	
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	

8 rows × 106 columns

In [19]:
```
datasets["application_train"].describe(include='all')
```

Out[19]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	A
count	307511.000000	307511.000000	307511	307511	307511	307511	307511.000000	
unique	NaN	NaN	2	3	2	2	NaN	
top	NaN	NaN	Cash loans	F	N	Y	NaN	
freq	NaN	NaN	278232	202448	202924	213312	NaN	
mean	278180.518577	0.080729	NaN	NaN	NaN	NaN	0.417052	
std	102790.175348	0.272419	NaN	NaN	NaN	NaN	0.722121	
min	100002.000000	0.000000	NaN	NaN	NaN	NaN	0.000000	
25%	189145.500000	0.000000	NaN	NaN	NaN	NaN	0.000000	
50%	278202.000000	0.000000	NaN	NaN	NaN	NaN	0.000000	
75%	367142.500000	0.000000	NaN	NaN	NaN	NaN	1.000000	
max	456255.000000	1.000000	NaN	NaN	NaN	NaN	19.000000	

11 rows × 122 columns

In [20]:
```
datasets["application_train"].corr()
```

Out[20]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOO
SK_ID_CURR	1.000000	-0.002108	-0.001129	-0.001820	-0.000343	-0.000433	
TARGET	-0.002108	1.000000	0.019187	-0.003982	-0.030369	-0.012817	
CNT_CHILDREN	-0.001129	0.019187	1.000000	0.012882	0.002145	0.021374	
AMT_INCOME_TOTAL	-0.001820	-0.003982	0.012882	1.000000	0.156870	0.191657	
AMT_CREDIT	-0.000343	-0.030369	0.002145	0.156870	1.000000	0.770138	
...	...	...	...	...	...	...	
AMT_REQ_CREDIT_BUREAU_DAY	-0.002193	0.002704	-0.000366	0.002944	0.004238	0.002185	
AMT_REQ_CREDIT_BUREAU_WEEK	0.002099	0.000788	-0.002436	0.002387	-0.001275	0.013881	
AMT_REQ_CREDIT_BUREAU_MON	0.000485	-0.012462	-0.010808	0.024700	0.054451	0.039148	
AMT_REQ_CREDIT_BUREAU_QRT	0.001025	-0.002022	-0.007836	0.004859	0.015925	0.010124	
AMT_REQ_CREDIT_BUREAU_YEAR	0.004659	0.019930	-0.041550	0.011690	-0.048448	-0.011320	

106 rows × 106 columns

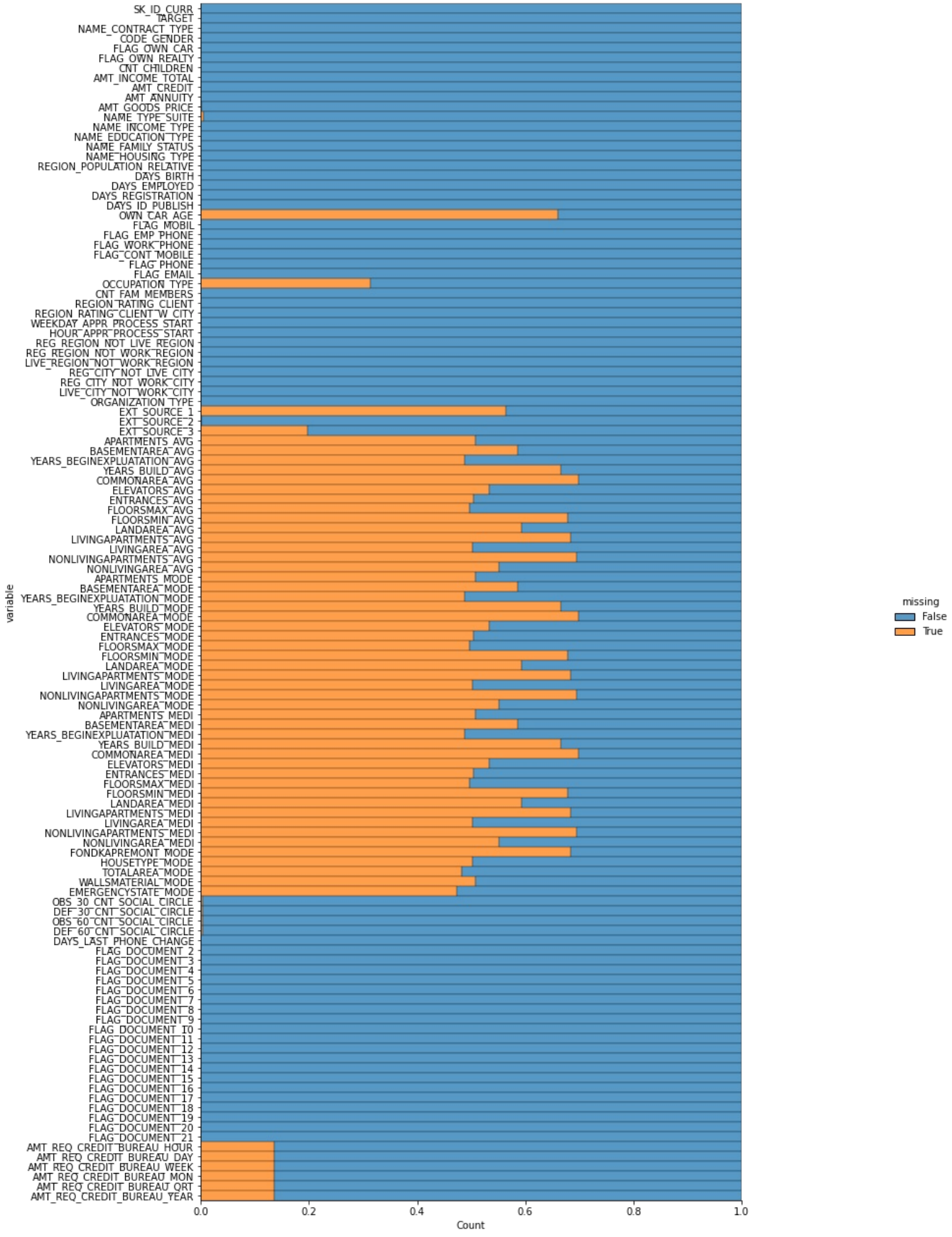
Missing values in Application Train

In [21]:
```
percent = (datasets["application_train"].isnull().sum()/datasets["application_train"].isnull().count()*100).sor
sum_missing = datasets["application_train"].isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Train Missing Cou
missing_application_train_data.head(20)
```

Out[21]:

	Percent	Train Missing Count
COMMONAREA_MEDI	69.87	214865
COMMONAREA_AVG	69.87	214865
COMMONAREA_MODE	69.87	214865
NONLIVINGAPARTMENTS_MODE	69.43	213514
NONLIVINGAPARTMENTS_AVG	69.43	213514
NONLIVINGAPARTMENTS_MEDI	69.43	213514
FONDKAPREMONT_MODE	68.39	210295
LIVINGAPARTMENTS_MODE	68.35	210199
LIVINGAPARTMENTS_AVG	68.35	210199
LIVINGAPARTMENTS_MEDI	68.35	210199
FLOORSMIN_AVG	67.85	208642
FLOORSMIN_MODE	67.85	208642
FLOORSMIN_MEDI	67.85	208642
YEARS_BUILD_MEDI	66.50	204488
YEARS_BUILD_MODE	66.50	204488
YEARS_BUILD_AVG	66.50	204488
OWN_CAR_AGE	65.99	202929
LANDAREA_MEDI	59.38	182590
LANDAREA_MODE	59.38	182590
LANDAREA_AVG	59.38	182590

In [22]:
```
plot_missing_data("application_train",18,20)
```



Summary of Application Test

In [23]:
```
datasets["application_test"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48744 entries, 0 to 48743
Columns: 121 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(40), object(16)
memory usage: 45.0+ MB
```

In [24]:
```
datasets["application_test"].columns
```

Out[24]:
```
Index(['SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR',
       'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT',
       'AMT_ANNUITY', 'AMT_GOODS_PRICE',
       ...
       'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20',
       'FLAG_DOCUMENT_21', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR'],
      dtype='object', length=121)
```

In [25]:
```
datasets["application_test"].dtypes
```

Out[25]:
```
SK_ID_CURR                     int64
NAME_CONTRACT_TYPE            object
CODE_GENDER                   object
FLAG_OWN_CAR                  object
FLAG_OWN_REALTY               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: 121, dtype: object
```

In [26]:
```
datasets["application_test"].describe() #numerical only features
```

Out[26]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_REL
count	48744.000000	48744.000000	4.874400e+04	4.874400e+04	48720.000000	4.874400e+04	48744.0
mean	277796.676350	0.397054	1.784318e+05	5.167404e+05	29426.240209	4.626188e+05	0.0
std	103169.547296	0.709047	1.015226e+05	3.653970e+05	16016.368315	3.367102e+05	0.0
min	100001.000000	0.000000	2.694150e+04	4.500000e+04	2295.000000	4.500000e+04	0.0
25%	188557.750000	0.000000	1.125000e+05	2.606400e+05	17973.000000	2.250000e+05	0.0
50%	277549.000000	0.000000	1.575000e+05	4.500000e+05	26199.000000	3.960000e+05	0.0
75%	367555.500000	1.000000	2.250000e+05	6.750000e+05	37390.500000	6.300000e+05	0.0
max	456250.000000	20.000000	4.410000e+06	2.245500e+06	180576.000000	2.245500e+06	0.0

8 rows × 105 columns

In [27]:
```
datasets["application_test"].describe(include='all') #look at all categorical and numerical
```

Out[27]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TO
count	48744.000000	48744	48744	48744	48744	48744.000000	4.874400e
unique	NaN	2	2	2	2	NaN	
top	NaN	Cash loans	F	N	Y	NaN	
freq	NaN	48305	32678	32311	33658	NaN	
mean	277796.676350	NaN	NaN	NaN	NaN	0.397054	1.784318e
std	103169.547296	NaN	NaN	NaN	NaN	0.709047	1.015226e
min	100001.000000	NaN	NaN	NaN	NaN	0.000000	2.694150e
25%	188557.750000	NaN	NaN	NaN	NaN	0.000000	1.125000e
50%	277549.000000	NaN	NaN	NaN	NaN	0.000000	1.575000e
75%	367555.500000	NaN	NaN	NaN	NaN	1.000000	2.250000e
max	456250.000000	NaN	NaN	NaN	NaN	20.000000	4.410000e

11 rows × 121 columns

In [28]:
```
datasets["application_test"].corr()
```

Out[28]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE
SK_ID_CURR	1.000000	0.000635	0.001278	0.005014	0.007112	0.005097
CNT_CHILDREN	0.000635	1.000000	0.038962	0.027840	0.056770	0.025507
AMT_INCOME_TOTAL	0.001278	0.038962	1.000000	0.396572	0.457833	0.401995
AMT_CREDIT	0.005014	0.027840	0.396572	1.000000	0.777733	0.988056
AMT_ANNUITY	0.007112	0.056770	0.457833	0.777733	1.000000	0.787033
...	...	...	...	...	...	...
AMT_REQ_CREDIT_BUREAU_DAY	0.001083	0.001539	0.004989	0.004882	0.006681	0.004865
AMT_REQ_CREDIT_BUREAU_WEEK	0.001178	0.007523	-0.002867	0.002904	0.003085	0.003358
AMT_REQ_CREDIT_BUREAU_MON	0.000430	-0.008337	0.008691	-0.000156	0.005695	-0.000254
AMT_REQ_CREDIT_BUREAU_QRT	-0.002092	0.029006	0.007410	-0.007750	0.012443	-0.008490
AMT_REQ_CREDIT_BUREAU_YEAR	0.003457	-0.039265	0.003281	-0.034533	-0.044901	-0.036227

105 rows × 105 columns

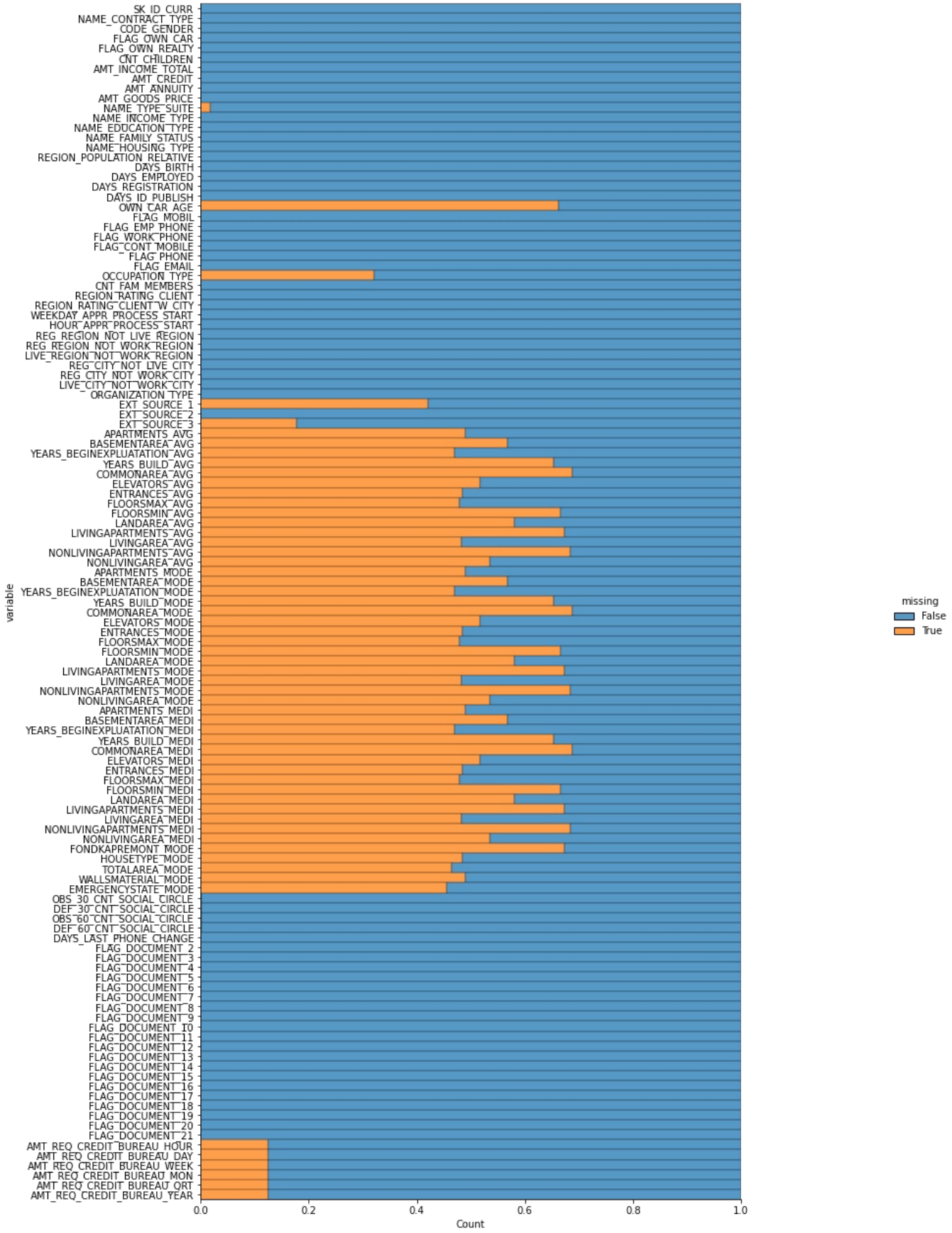
Missing data for Application Test

In [29]:
```
percent = (datasets["application_test"].isnull().sum()/datasets["application_test"].isnull().count()*100).sort_
sum_missing = datasets["application_test"].isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Train Missing Cou
missing_application_train_data.head(20)
```

Out[29]:

	Percent	Train Missing Count
COMMONAREA_AVG	68.72	33495
COMMONAREA_MODE	68.72	33495
COMMONAREA_MEDI	68.72	33495
NONLIVINGAPARTMENTS_AVG	68.41	33347
NONLIVINGAPARTMENTS_MODE	68.41	33347
NONLIVINGAPARTMENTS_MEDI	68.41	33347
FONDKAPREMONT_MODE	67.28	32797
LIVINGAPARTMENTS_AVG	67.25	32780
LIVINGAPARTMENTS_MODE	67.25	32780
LIVINGAPARTMENTS_MEDI	67.25	32780
FLOORSMIN_MEDI	66.61	32466
FLOORSMIN_AVG	66.61	32466
FLOORSMIN_MODE	66.61	32466
OWN_CAR_AGE	66.29	32312
YEARS_BUILD_AVG	65.28	31818
YEARS_BUILD_MEDI	65.28	31818
YEARS_BUILD_MODE	65.28	31818
LANDAREA_MEDI	57.96	28254
LANDAREA_AVG	57.96	28254
LANDAREA_MODE	57.96	28254

In [30]:
```
plot_missing_data("application_test",18,20)
```



Summary of Bureau

In [31]:
```
datasets["bureau"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1716428 entries, 0 to 1716427
Data columns (total 17 columns):
 #   Column                  Dtype  
---  ------                  -----  
 0   SK_ID_CURR              int64  
 1   SK_ID_BUREAU            int64  
 2   CREDIT_ACTIVE           object 
 3   CREDIT_CURRENCY         object 
 4   DAYS_CREDIT             int64  
 5   CREDIT_DAY_OVERDUE      int64  
 6   DAYS_CREDIT_ENDDATE     float64
 7   DAYS_ENDDATE_FACT       float64
 8   AMT_CREDIT_MAX_OVERDUE  float64
 9   CNT_CREDIT_PROLONG      int64  
 10  AMT_CREDIT_SUM          float64
 11  AMT_CREDIT_SUM_DEBT     float64
 12  AMT_CREDIT_SUM_LIMIT    float64
 13  AMT_CREDIT_SUM_OVERDUE  float64
 14  CREDIT_TYPE             object 
 15  DAYS_CREDIT_UPDATE      int64  
 16  AMT_ANNUITY             float64
dtypes: float64(8), int64(6), object(3)
memory usage: 222.6+ MB
```

In [32]:
```
datasets["bureau"].columns
```

Out[32]:
```
Index(['SK_ID_CURR', 'SK_ID_BUREAU', 'CREDIT_ACTIVE', 'CREDIT_CURRENCY',
       'DAYS_CREDIT', 'CREDIT_DAY_OVERDUE', 'DAYS_CREDIT_ENDDATE',
       'DAYS_ENDDATE_FACT', 'AMT_CREDIT_MAX_OVERDUE', 'CNT_CREDIT_PROLONG',
       'AMT_CREDIT_SUM', 'AMT_CREDIT_SUM_DEBT', 'AMT_CREDIT_SUM_LIMIT',
       'AMT_CREDIT_SUM_OVERDUE', 'CREDIT_TYPE', 'DAYS_CREDIT_UPDATE',
       'AMT_ANNUITY'],
      dtype='object')
```

In [33]:
```
datasets["bureau"].dtypes
```

Out[33]:
```
SK_ID_CURR                 int64
SK_ID_BUREAU               int64
CREDIT_ACTIVE             object
CREDIT_CURRENCY           object
DAYS_CREDIT                int64
CREDIT_DAY_OVERDUE         int64
DAYS_CREDIT_ENDDATE      float64
DAYS_ENDDATE_FACT        float64
AMT_CREDIT_MAX_OVERDUE   float64
CNT_CREDIT_PROLONG         int64
AMT_CREDIT_SUM           float64
AMT_CREDIT_SUM_DEBT      float64
AMT_CREDIT_SUM_LIMIT     float64
AMT_CREDIT_SUM_OVERDUE   float64
CREDIT_TYPE               object
DAYS_CREDIT_UPDATE         int64
AMT_ANNUITY              float64
dtype: object
```

In [34]:
```
datasets["bureau"].describe()
```

Out[34]:

	SK_ID_CURR	SK_ID_BUREAU	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	DAYS_ENDDATE_FACT	AMT_CREDI
count	1.716428e+06	1.716428e+06	1.716428e+06	1.716428e+06	1.610875e+06	1.082775e+06	
mean	2.782149e+05	5.924434e+06	-1.142108e+03	8.181666e-01	5.105174e+02	-1.017437e+03	
std	1.029386e+05	5.322657e+05	7.951649e+02	3.654443e+01	4.994220e+03	7.140106e+02	
min	1.000010e+05	5.000000e+06	-2.922000e+03	0.000000e+00	-4.206000e+04	-4.202300e+04	
25%	1.888668e+05	5.463954e+06	-1.666000e+03	0.000000e+00	-1.138000e+03	-1.489000e+03	
50%	2.780550e+05	5.926304e+06	-9.870000e+02	0.000000e+00	-3.300000e+02	-8.970000e+02	
75%	3.674260e+05	6.385681e+06	-4.740000e+02	0.000000e+00	4.740000e+02	-4.250000e+02	
max	4.562550e+05	6.843457e+06	0.000000e+00	2.792000e+03	3.119900e+04	0.000000e+00	

In [35]:
```
datasets["bureau"].describe(include='all')
```

Out[35]:

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDA
count	1.716428e+06	1.716428e+06	1716428	1716428	1.716428e+06	1.716428e+06	1.610875e+
unique	NaN	NaN	4	4	NaN	NaN	N
top	NaN	NaN	Closed	currency 1	NaN	NaN	N
freq	NaN	NaN	1079273	1715020	NaN	NaN	N
mean	2.782149e+05	5.924434e+06	NaN	NaN	-1.142108e+03	8.181666e-01	5.105174e+
std	1.029386e+05	5.322657e+05	NaN	NaN	7.951649e+02	3.654443e+01	4.994220e+
min	1.000010e+05	5.000000e+06	NaN	NaN	-2.922000e+03	0.000000e+00	-4.206000e+
25%	1.888668e+05	5.463954e+06	NaN	NaN	-1.666000e+03	0.000000e+00	-1.138000e+
50%	2.780550e+05	5.926304e+06	NaN	NaN	-9.870000e+02	0.000000e+00	-3.300000e+
75%	3.674260e+05	6.385681e+06	NaN	NaN	-4.740000e+02	0.000000e+00	4.740000e+
max	4.562550e+05	6.843457e+06	NaN	NaN	0.000000e+00	2.792000e+03	3.119900e+

In [36]:
```
datasets["bureau"].corr()
```

Out[36]:

	SK_ID_CURR	SK_ID_BUREAU	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	DAYS_ENDD
SK_ID_CURR	1.000000	0.000135	0.000266	0.000283	0.000456	
SK_ID_BUREAU	0.000135	1.000000	0.013015	-0.002628	0.009107	
DAYS_CREDIT	0.000266	0.013015	1.000000	-0.027266	0.225682	
CREDIT_DAY_OVERDUE	0.000283	-0.002628	-0.027266	1.000000	-0.007352	
DAYS_CREDIT_ENDDATE	0.000456	0.009107	0.225682	-0.007352	1.000000	
DAYS_ENDDATE_FACT	-0.000648	0.017890	0.875359	-0.008637	0.248825	
AMT_CREDIT_MAX_OVERDUE	0.001329	0.002290	-0.014724	0.001249	0.000577	
CNT_CREDIT_PROLONG	-0.000388	-0.000740	-0.030460	0.002756	0.113683	
AMT_CREDIT_SUM	0.001179	0.007962	0.050883	-0.003292	0.055424	
AMT_CREDIT_SUM_DEBT	-0.000790	0.005732	0.135397	-0.002355	0.081298	
AMT_CREDIT_SUM_LIMIT	-0.000304	-0.003986	0.025140	-0.000345	0.095421	
AMT_CREDIT_SUM_OVERDUE	-0.000014	-0.000499	-0.000383	0.090951	0.001077	
DAYS_CREDIT_UPDATE	0.000510	0.019398	0.688771	-0.018461	0.248525	
AMT_ANNUITY	-0.002727	0.001799	0.005676	-0.000339	0.000475	

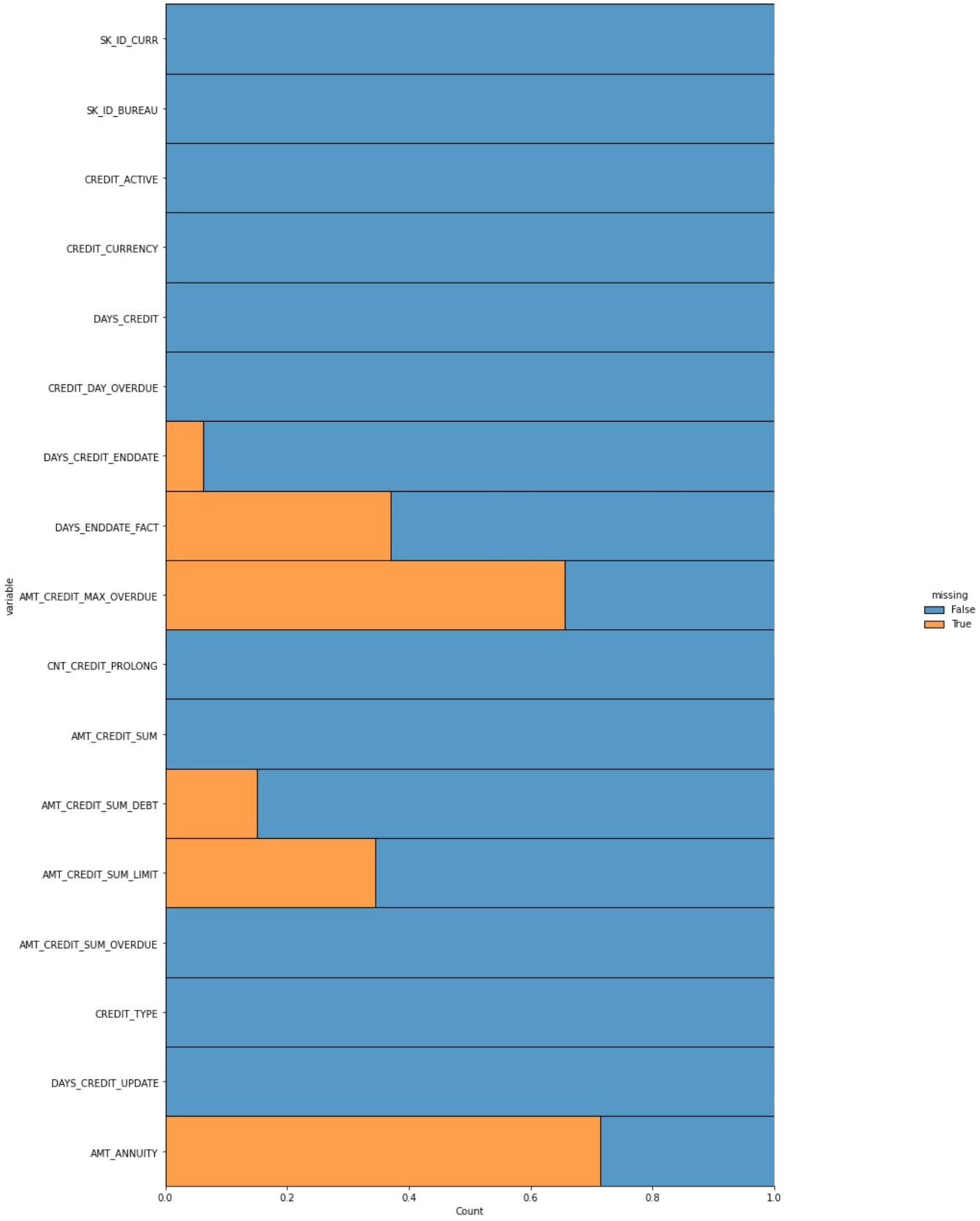
Missing data for Bureau

In [37]:
```
percent = (datasets["bureau"].isnull().sum()/datasets["bureau"].isnull().count()*100).sort_values(ascending = F
sum_missing = datasets["bureau"].isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Test Missing Coun
missing_application_train_data.head(20)
```

Out[37]:

	Percent	Test Missing Count
AMT_ANNUITY	71.47	1226791
AMT_CREDIT_MAX_OVERDUE	65.51	1124488
DAYS_ENDDATE_FACT	36.92	633653
AMT_CREDIT_SUM_LIMIT	34.48	591780
AMT_CREDIT_SUM_DEBT	15.01	257669
DAYS_CREDIT_ENDDATE	6.15	105553
AMT_CREDIT_SUM	0.00	13
CREDIT_ACTIVE	0.00	0
CREDIT_CURRENCY	0.00	0
DAYS_CREDIT	0.00	0
CREDIT_DAY_OVERDUE	0.00	0
SK_ID_BUREAU	0.00	0
CNT_CREDIT_PROLONG	0.00	0
AMT_CREDIT_SUM_OVERDUE	0.00	0
CREDIT_TYPE	0.00	0
DAYS_CREDIT_UPDATE	0.00	0
SK_ID_CURR	0.00	0

In [38]:
```
plot_missing_data("bureau",18,20)
```



Summary of Bureau Balance

In [8]:
```
datasets["bureau_balance"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27299925 entries, 0 to 27299924
Data columns (total 3 columns):
 #   Column          Dtype
---  ------          -----
 0   SK_ID_BUREAU    int64
 1   MONTHS_BALANCE  int64
 2   STATUS          object
dtypes: int64(2), object(1)
memory usage: 624.8+ MB
```

In [9]:
```
datasets["bureau_balance"].columns
```

Out[9]:
```
Index(['SK_ID_BUREAU', 'MONTHS_BALANCE', 'STATUS'], dtype='object')
```

In [10]:
```
datasets["bureau_balance"].dtypes
```

Out[10]:
```
SK_ID_BUREAU       int64
MONTHS_BALANCE     int64
STATUS            object
dtype: object
```

In [11]:
```
datasets["bureau_balance"].describe()
```

Out[11]:

	SK_ID_BUREAU	MONTHS_BALANCE
count	2.729992e+07	2.729992e+07
mean	6.036297e+06	-3.074169e+01
std	4.923489e+05	2.386451e+01
min	5.001709e+06	-9.600000e+01
25%	5.730933e+06	-4.600000e+01
50%	6.070821e+06	-2.500000e+01
75%	6.431951e+06	-1.100000e+01
max	6.842888e+06	0.000000e+00

In [12]:
```
datasets["bureau_balance"].describe(include='all')
```

Out[12]:

	SK_ID_BUREAU	MONTHS_BALANCE	STATUS
count	2.729992e+07	2.729992e+07	27299925
unique	NaN	NaN	8
top	NaN	NaN	C
freq	NaN	NaN	13646993
mean	6.036297e+06	-3.074169e+01	NaN
std	4.923489e+05	2.386451e+01	NaN
min	5.001709e+06	-9.600000e+01	NaN
25%	5.730933e+06	-4.600000e+01	NaN
50%	6.070821e+06	-2.500000e+01	NaN
75%	6.431951e+06	-1.100000e+01	NaN
max	6.842888e+06	0.000000e+00	NaN

In [13]:
```
datasets["bureau_balance"].corr()
```

Out[13]:

	SK_ID_BUREAU	MONTHS_BALANCE
SK_ID_BUREAU	1.000000	0.011873
MONTHS_BALANCE	0.011873	1.000000

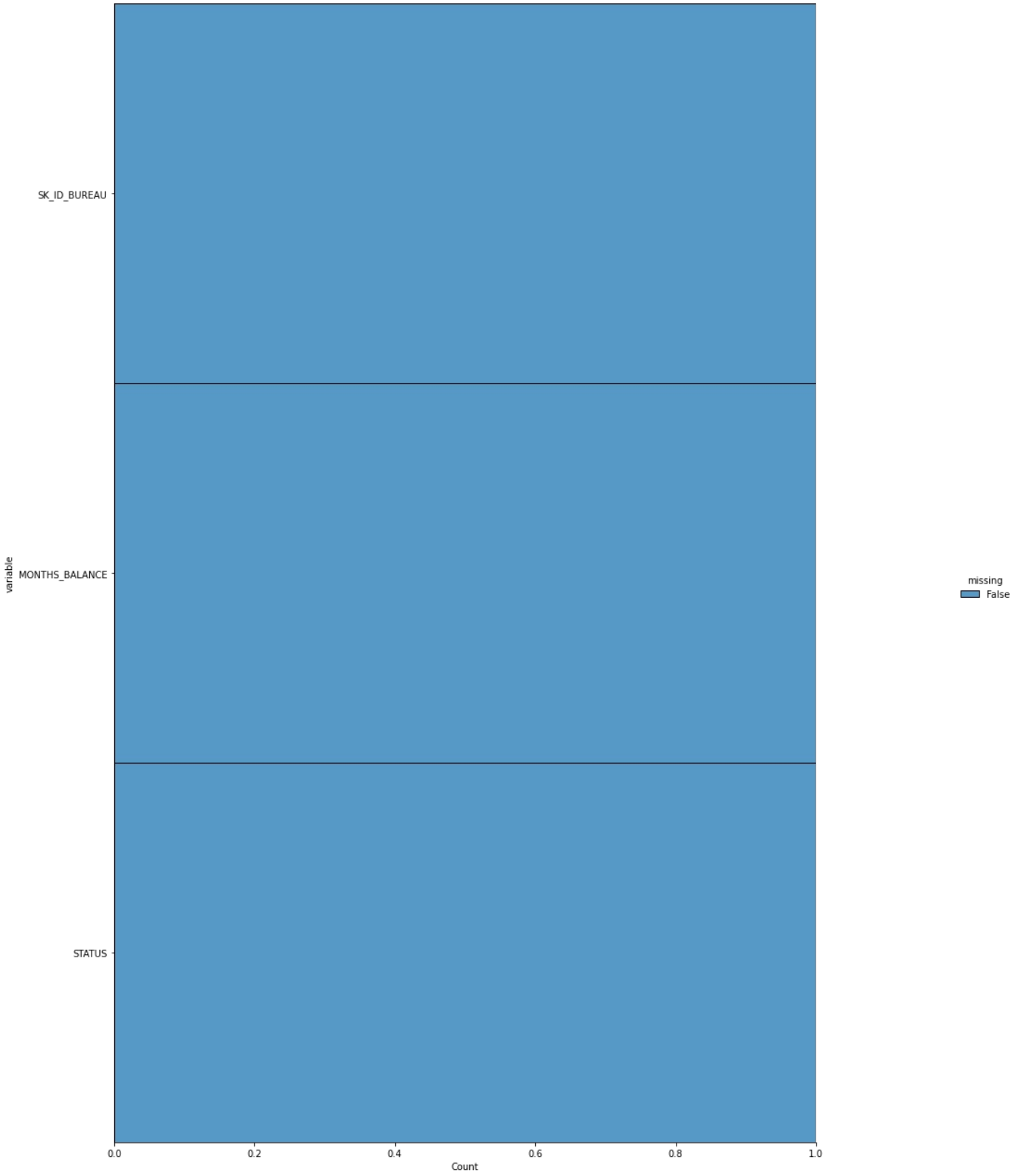
Missing data for Bureau Balance

In [14]:
```
percent = (datasets["bureau_balance"].isnull().sum()/datasets["bureau_balance"].isnull().count()*100).sort_valu
sum_missing = datasets["bureau_balance"].isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Test Missing Coun
missing_application_train_data.head(20)
```

Out[14]:

	Percent	Test Missing Count
SK_ID_BUREAU	0.0	0
MONTHS_BALANCE	0.0	0
STATUS	0.0	0

In [15]:
```
plot_missing_data("bureau_balance",18,20)
```



Summary of POS_CASH_balance

In [6]:
```
datasets["POS_CASH_balance"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3829580 entries, 0 to 3829579
Data columns (total 8 columns):
 #   Column                 Dtype
---  ------                 -----
 0   SK_ID_PREV             int64
 1   SK_ID_CURR             int64
 2   MONTHS_BALANCE         int64
 3   CNT_INSTALMENT         float64
 4   CNT_INSTALMENT_FUTURE  float64
 5   NAME_CONTRACT_STATUS   object
 6   SK_DPD                 float64
 7   SK_DPD_DEF             float64
dtypes: float64(4), int64(3), object(1)
memory usage: 233.7+ MB
```

In [7]:
```
datasets["POS_CASH_balance"].columns
```

Out[7]:
```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'MONTHS_BALANCE', 'CNT_INSTALMENT',
       'CNT_INSTALMENT_FUTURE', 'NAME_CONTRACT_STATUS', 'SK_DPD',
       'SK_DPD_DEF'],
      dtype='object')
```

In [8]:
```
datasets["POS_CASH_balance"].dtypes
```

Out[8]:
```
SK_ID_PREV                int64
SK_ID_CURR                int64
MONTHS_BALANCE            int64
CNT_INSTALMENT          float64
CNT_INSTALMENT_FUTURE   float64
NAME_CONTRACT_STATUS     object
SK_DPD                  float64
SK_DPD_DEF              float64
dtype: object
```

In [9]:
```
datasets["POS_CASH_balance"].describe()
```

Out[9]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FUTURE	SK_DPD	SK_DPD_DEF
count	3.829580e+06	3.829580e+06	3.829580e+06	3.823444e+06	3.823437e+06	3.829579e+06	3.829579e+06
mean	1.904375e+06	2.785338e+05	-3.214404e+01	1.956578e+01	1.283459e+01	4.358176e-01	7.258109e-02
std	5.355338e+05	1.027329e+05	2.549135e+01	1.380046e+01	1.273046e+01	1.744642e+01	1.541065e+00
min	1.000001e+06	1.000010e+05	-9.600000e+01	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	1.435030e+06	1.896800e+05	-4.600000e+01	1.000000e+01	4.000000e+00	0.000000e+00	0.000000e+00
50%	1.898227e+06	2.788660e+05	-2.300000e+01	1.200000e+01	9.000000e+00	0.000000e+00	0.000000e+00
75%	2.369573e+06	3.676380e+05	-1.200000e+01	2.400000e+01	1.800000e+01	0.000000e+00	0.000000e+00
max	2.843499e+06	4.562550e+05	-1.000000e+00	9.200000e+01	8.500000e+01	3.006000e+03	4.190000e+02

In [10]:
```
datasets["POS_CASH_balance"].describe(include='all')
```

Out[10]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FUTURE	NAME_CONTRACT_STATUS	SI
count	3.829580e+06	3.829580e+06	3.829580e+06	3.823444e+06	3.823437e+06	3829579	3.8295
unique	NaN	NaN	NaN	NaN	NaN	8	
top	NaN	NaN	NaN	NaN	NaN	Active	
freq	NaN	NaN	NaN	NaN	NaN	3570142	
mean	1.904375e+06	2.785338e+05	-3.214404e+01	1.956578e+01	1.283459e+01	NaN	4.3581
std	5.355338e+05	1.027329e+05	2.549135e+01	1.380046e+01	1.273046e+01	NaN	1.7446
min	1.000001e+06	1.000010e+05	-9.600000e+01	1.000000e+00	0.000000e+00	NaN	0.0000
25%	1.435030e+06	1.896800e+05	-4.600000e+01	1.000000e+01	4.000000e+00	NaN	0.0000
50%	1.898227e+06	2.788660e+05	-2.300000e+01	1.200000e+01	9.000000e+00	NaN	0.0000
75%	2.369573e+06	3.676380e+05	-1.200000e+01	2.400000e+01	1.800000e+01	NaN	0.0000
max	2.843499e+06	4.562550e+05	-1.000000e+00	9.200000e+01	8.500000e+01	NaN	3.0060

In [11]:
```
datasets["POS_CASH_balance"].corr()
```

Out[11]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FUTURE	SK_DPD	SK_D
SK_ID_PREV	1.000000	-0.000208	0.003497	0.003542	0.003431	0.000632	
SK_ID_CURR	-0.000208	1.000000	0.000430	0.000618	-0.000105	-0.000401	
MONTHS_BALANCE	0.003497	0.000430	1.000000	0.433006	0.351605	-0.010548	
CNT_INSTALMENT	0.003542	0.000618	0.433006	1.000000	0.897199	-0.013366	
CNT_INSTALMENT_FUTURE	0.003431	-0.000105	0.351605	0.897199	1.000000	-0.020738	
SK_DPD	0.000632	-0.000401	-0.010548	-0.013366	-0.020738	1.000000	
SK_DPD_DEF	0.000186	0.002109	-0.027817	-0.009263	-0.017952	0.090650	

Missing data for POS_CASH_balance

In [12]:
```
percent = (datasets["POS_CASH_balance"].isnull().sum()/datasets["POS_CASH_balance"].isnull().count()*100).sort_
sum_missing = datasets["POS_CASH_balance"].isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Test Missing Coun
missing_application_train_data.head(20)
```

Out[12]:

	Percent	Test Missing Count
CNT_INSTALMENT_FUTURE	0.16	6143
CNT_INSTALMENT	0.16	6136
NAME_CONTRACT_STATUS	0.00	1
SK_DPD	0.00	1
SK_DPD_DEF	0.00	1
SK_ID_PREV	0.00	0
SK_ID_CURR	0.00	0
MONTHS_BALANCE	0.00	0

In [ ]:
```
plot_missing_data("POS_CASH_balance",18,20)
```

Summary of credit_card_balance

In [13]:
```
datasets["credit_card_balance"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3840312 entries, 0 to 3840311
Data columns (total 23 columns):
 #   Column                      Dtype  
---  ------                      -----  
 0   SK_ID_PREV                  int64  
 1   SK_ID_CURR                  int64  
 2   MONTHS_BALANCE              int64  
 3   AMT_BALANCE                 float64
 4   AMT_CREDIT_LIMIT_ACTUAL     int64  
 5   AMT_DRAWINGS_ATM_CURRENT    float64
 6   AMT_DRAWINGS_CURRENT        float64
 7   AMT_DRAWINGS_OTHER_CURRENT  float64
 8   AMT_DRAWINGS_POS_CURRENT    float64
 9   AMT_INST_MIN_REGULARITY     float64
 10  AMT_PAYMENT_CURRENT         float64
 11  AMT_PAYMENT_TOTAL_CURRENT   float64
 12  AMT_RECEIVABLE_PRINCIPAL    float64
 13  AMT_RECIVABLE               float64
 14  AMT_TOTAL_RECEIVABLE        float64
 15  CNT_DRAWINGS_ATM_CURRENT    float64
 16  CNT_DRAWINGS_CURRENT        int64  
 17  CNT_DRAWINGS_OTHER_CURRENT  float64
 18  CNT_DRAWINGS_POS_CURRENT    float64
 19  CNT_INSTALMENT_MATURE_CUM   float64
 20  NAME_CONTRACT_STATUS        object 
 21  SK_DPD                      int64  
 22  SK_DPD_DEF                  int64  
dtypes: float64(15), int64(7), object(1)
memory usage: 673.9+ MB
```

In [14]:
```
datasets["credit_card_balance"].columns
```

Out[14]:
```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'MONTHS_BALANCE', 'AMT_BALANCE',
       'AMT_CREDIT_LIMIT_ACTUAL', 'AMT_DRAWINGS_ATM_CURRENT',
       'AMT_DRAWINGS_CURRENT', 'AMT_DRAWINGS_OTHER_CURRENT',
       'AMT_DRAWINGS_POS_CURRENT', 'AMT_INST_MIN_REGULARITY',
       'AMT_PAYMENT_CURRENT', 'AMT_PAYMENT_TOTAL_CURRENT',
       'AMT_RECEIVABLE_PRINCIPAL', 'AMT_RECIVABLE', 'AMT_TOTAL_RECEIVABLE',
       'CNT_DRAWINGS_ATM_CURRENT', 'CNT_DRAWINGS_CURRENT',
       'CNT_DRAWINGS_OTHER_CURRENT', 'CNT_DRAWINGS_POS_CURRENT',
       'CNT_INSTALMENT_MATURE_CUM', 'NAME_CONTRACT_STATUS', 'SK_DPD',
       'SK_DPD_DEF'],
      dtype='object')
```

In [15]:
```
datasets["credit_card_balance"].dtypes
```

Out[15]:
```
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
SK_DPD                       int64
SK_DPD_DEF                   int64
dtype: object
```

In [16]:
```
datasets["credit_card_balance"].describe()
```

Out[16]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_DRAWINGS_ATM_CURRENT	AMT
count	3.840312e+06	3.840312e+06	3.840312e+06	3.840312e+06	3.840312e+06	3.090496e+06	
mean	1.904504e+06	2.783242e+05	-3.452192e+01	5.830016e+04	1.538080e+05	5.961325e+03	
std	5.364695e+05	1.027045e+05	2.666775e+01	1.063070e+05	1.651457e+05	2.822569e+04	
min	1.000018e+06	1.000060e+05	-9.600000e+01	-4.202502e+05	0.000000e+00	-6.827310e+03	
25%	1.434385e+06	1.895170e+05	-5.500000e+01	0.000000e+00	4.500000e+04	0.000000e+00	
50%	1.897122e+06	2.783960e+05	-2.800000e+01	0.000000e+00	1.125000e+05	0.000000e+00	
75%	2.369328e+06	3.675800e+05	-1.100000e+01	8.904669e+04	1.800000e+05	0.000000e+00	
max	2.843496e+06	4.562500e+05	-1.000000e+00	1.505902e+06	1.350000e+06	2.115000e+06	

8 rows × 22 columns

In [17]:
```
datasets["credit_card_balance"].describe(include='all')
```

Out[17]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_DRAWINGS_ATM_CURRENT	AM
count	3.840312e+06	3.840312e+06	3.840312e+06	3.840312e+06	3.840312e+06	3.090496e+06	
unique	NaN	NaN	NaN	NaN	NaN	NaN	
top	NaN	NaN	NaN	NaN	NaN	NaN	
freq	NaN	NaN	NaN	NaN	NaN	NaN	
mean	1.904504e+06	2.783242e+05	-3.452192e+01	5.830016e+04	1.538080e+05	5.961325e+03	
std	5.364695e+05	1.027045e+05	2.666775e+01	1.063070e+05	1.651457e+05	2.822569e+04	
min	1.000018e+06	1.000060e+05	-9.600000e+01	-4.202502e+05	0.000000e+00	-6.827310e+03	
25%	1.434385e+06	1.895170e+05	-5.500000e+01	0.000000e+00	4.500000e+04	0.000000e+00	
50%	1.897122e+06	2.783960e+05	-2.800000e+01	0.000000e+00	1.125000e+05	0.000000e+00	
75%	2.369328e+06	3.675800e+05	-1.100000e+01	8.904669e+04	1.800000e+05	0.000000e+00	
max	2.843496e+06	4.562500e+05	-1.000000e+00	1.505902e+06	1.350000e+06	2.115000e+06	

11 rows × 23 columns

In [18]:
```
datasets["credit_card_balance"].corr()
```

Out[18]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_DRA
SK_ID_PREV	1.000000	0.004723	0.003670	0.005046	0.006631	
SK_ID_CURR	0.004723	1.000000	0.001696	0.003510	0.005991	
MONTHS_BALANCE	0.003670	0.001696	1.000000	0.014558	0.199900	
AMT_BALANCE	0.005046	0.003510	0.014558	1.000000	0.489386	
AMT_CREDIT_LIMIT_ACTUAL	0.006631	0.005991	0.199900	0.489386	1.000000	
AMT_DRAWINGS_ATM_CURRENT	0.004342	0.000814	0.036802	0.283551	0.247219	
AMT_DRAWINGS_CURRENT	0.002624	0.000708	0.065527	0.336965	0.263093	
AMT_DRAWINGS_OTHER_CURRENT	-0.000160	0.000958	0.000405	0.065366	0.050579	
AMT_DRAWINGS_POS_CURRENT	0.001721	-0.000786	0.118146	0.169449	0.234976	
AMT_INST_MIN_REGULARITY	0.006460	0.003300	-0.087529	0.896728	0.467620	
AMT_PAYMENT_CURRENT	0.003472	0.000127	0.076355	0.143934	0.308294	
AMT_PAYMENT_TOTAL_CURRENT	0.001641	0.000784	0.035614	0.151349	0.226570	
AMT_RECEIVABLE_PRINCIPAL	0.005140	0.003589	0.016266	0.999720	0.490445	
AMT_RECIVABLE	0.005035	0.003518	0.013172	0.999917	0.488641	
AMT_TOTAL_RECEIVABLE	0.005032	0.003524	0.013084	0.999897	0.488598	
CNT_DRAWINGS_ATM_CURRENT	0.002821	0.002082	0.002536	0.309968	0.221808	
CNT_DRAWINGS_CURRENT	0.000367	0.002654	0.113321	0.259184	0.204237	
CNT_DRAWINGS_OTHER_CURRENT	-0.001412	-0.000131	-0.026192	0.046563	0.030051	
CNT_DRAWINGS_POS_CURRENT	0.000809	0.002135	0.160207	0.155553	0.202868	
CNT_INSTALMENT_MATURE_CUM	-0.007219	-0.000581	-0.008620	0.005009	-0.157269	
SK_DPD	-0.001786	-0.000962	0.039434	-0.046988	-0.038791	
SK_DPD_DEF	0.001973	0.001519	0.001659	0.013009	-0.002236	

22 rows × 22 columns

Missing data for credit_card_balance

In [19]:
```
percent = (datasets["credit_card_balance"].isnull().sum()/datasets["credit_card_balance"].isnull().count()*100)
sum_missing = datasets["credit_card_balance"].isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Test Missing Coun
missing_application_train_data.head(20)
```

Out[19]:

	Percent	Test Missing Count
AMT_PAYMENT_CURRENT	20.00	767988
AMT_DRAWINGS_ATM_CURRENT	19.52	749816
CNT_DRAWINGS_POS_CURRENT	19.52	749816
AMT_DRAWINGS_OTHER_CURRENT	19.52	749816
AMT_DRAWINGS_POS_CURRENT	19.52	749816
CNT_DRAWINGS_OTHER_CURRENT	19.52	749816
CNT_DRAWINGS_ATM_CURRENT	19.52	749816
CNT_INSTALMENT_MATURE_CUM	7.95	305236
AMT_INST_MIN_REGULARITY	7.95	305236
SK_ID_PREV	0.00	0
AMT_TOTAL_RECEIVABLE	0.00	0
SK_DPD	0.00	0
NAME_CONTRACT_STATUS	0.00	0
CNT_DRAWINGS_CURRENT	0.00	0
AMT_PAYMENT_TOTAL_CURRENT	0.00	0
AMT_RECIVABLE	0.00	0
AMT_RECEIVABLE_PRINCIPAL	0.00	0
SK_ID_CURR	0.00	0
AMT_DRAWINGS_CURRENT	0.00	0
AMT_CREDIT_LIMIT_ACTUAL	0.00	0

In [ ]:
```
plot_missing_data("credit_card_balance",18,20)
```

Summary of previous_application

In [20]:
```
datasets["previous_application"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 37 columns):
 #   Column                       Non-Null Count    Dtype  
---  ------                       --------------    -----  
 0   SK_ID_PREV                   1670214 non-null  int64  
 1   SK_ID_CURR                   1670214 non-null  int64  
 2   NAME_CONTRACT_TYPE           1670214 non-null  object 
 3   AMT_ANNUITY                  1297979 non-null  float64
 4   AMT_APPLICATION              1670214 non-null  float64
 5   AMT_CREDIT                   1670213 non-null  float64
 6   AMT_DOWN_PAYMENT             774370 non-null   float64
 7   AMT_GOODS_PRICE              1284699 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object 
 9   HOUR_APPR_PROCESS_START      1670214 non-null  int64  
 10  FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object 
 11  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64  
 12  RATE_DOWN_PAYMENT            774370 non-null   float64
 13  RATE_INTEREST_PRIMARY        5951 non-null     float64
 14  RATE_INTEREST_PRIVILEGED     5951 non-null     float64
 15  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object 
 16  NAME_CONTRACT_STATUS         1670214 non-null  object 
 17  DAYS_DECISION                1670214 non-null  int64  
 18  NAME_PAYMENT_TYPE            1670214 non-null  object 
 19  CODE_REJECT_REASON           1670214 non-null  object 
 20  NAME_TYPE_SUITE              849809 non-null   object 
 21  NAME_CLIENT_TYPE             1670214 non-null  object 
 22  NAME_GOODS_CATEGORY          1670214 non-null  object 
 23  NAME_PORTFOLIO               1670214 non-null  object 
 24  NAME_PRODUCT_TYPE            1670214 non-null  object 
 25  CHANNEL_TYPE                 1670214 non-null  object 
 26  SELLERPLACE_AREA             1670214 non-null  int64  
 27  NAME_SELLER_INDUSTRY         1670214 non-null  object 
 28  CNT_PAYMENT                  1297984 non-null  float64
 29  NAME_YIELD_GROUP             1670214 non-null  object 
 30  PRODUCT_COMBINATION          1669868 non-null  object 
 31  DAYS_FIRST_DRAWING           997149 non-null   float64
 32  DAYS_FIRST_DUE               997149 non-null   float64
 33  DAYS_LAST_DUE_1ST_VERSION    997149 non-null   float64
 34  DAYS_LAST_DUE                997149 non-null   float64
 35  DAYS_TERMINATION             997149 non-null   float64
 36  NFLAG_INSURED_ON_APPROVAL    997149 non-null   float64
dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
```

In [21]:
```
datasets["previous_application"].columns
```

Out[21]:
```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY',
       'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRICE',
       'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START',
       'FLAG_LAST_APPL_PER_CONTRACT', 'NFLAG_LAST_APPL_IN_DAY',
       'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY',
       'RATE_INTEREST_PRIVILEGED', 'NAME_CASH_LOAN_PURPOSE',
       'NAME_CONTRACT_STATUS', 'DAYS_DECISION', 'NAME_PAYMENT_TYPE',
       'CODE_REJECT_REASON', 'NAME_TYPE_SUITE', 'NAME_CLIENT_TYPE',
       'NAME_GOODS_CATEGORY', 'NAME_PORTFOLIO', 'NAME_PRODUCT_TYPE',
       'CHANNEL_TYPE', 'SELLERPLACE_AREA', 'NAME_SELLER_INDUSTRY',
       'CNT_PAYMENT', 'NAME_YIELD_GROUP', 'PRODUCT_COMBINATION',
       'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION',
       'DAYS_LAST_DUE', 'DAYS_TERMINATION', 'NFLAG_INSURED_ON_APPROVAL'],
      dtype='object')
```

In [22]:
```
datasets["previous_application"].dtypes
```

Out[22]:
```
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
NAME_CLIENT_TYPE             object
NAME_GOODS_CATEGORY          object
NAME_PORTFOLIO               object
NAME_PRODUCT_TYPE            object
CHANNEL_TYPE                 object
SELLERPLACE_AREA              int64
NAME_SELLER_INDUSTRY         object
CNT_PAYMENT                 float64
NAME_YIELD_GROUP             object
PRODUCT_COMBINATION          object
DAYS_FIRST_DRAWING          float64
DAYS_FIRST_DUE              float64
DAYS_LAST_DUE_1ST_VERSION   float64
DAYS_LAST_DUE               float64
DAYS_TERMINATION            float64
NFLAG_INSURED_ON_APPROVAL   float64
dtype: object
```

In [23]:
```
datasets["previous_application"].describe()
```

Out[23]:

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	HOUR_AI
count	1.670214e+06	1.670214e+06	1.297979e+06	1.670214e+06	1.670213e+06	7.743700e+05	1.284699e+06	
mean	1.923089e+06	2.783572e+05	1.595512e+04	1.752339e+05	1.961140e+05	6.697402e+03	2.278473e+05	
std	5.325980e+05	1.028148e+05	1.478214e+04	2.927798e+05	3.185746e+05	2.092150e+04	3.153966e+05	
min	1.000001e+06	1.000010e+05	0.000000e+00	0.000000e+00	0.000000e+00	-9.000000e-01	0.000000e+00	
25%	1.461857e+06	1.893290e+05	6.321780e+03	1.872000e+04	2.416050e+04	0.000000e+00	5.084100e+04	
50%	1.923110e+06	2.787145e+05	1.125000e+04	7.104600e+04	8.054100e+04	1.638000e+03	1.123200e+05	
75%	2.384280e+06	3.675140e+05	2.065842e+04	1.803600e+05	2.164185e+05	7.740000e+03	2.340000e+05	
max	2.845382e+06	4.562550e+05	4.180581e+05	6.905160e+06	6.905160e+06	3.060045e+06	6.905160e+06	

8 rows × 21 columns

In [24]:
```
datasets["previous_application"].describe(include='all')
```

Out[24]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AM
count	1.670214e+06	1.670214e+06	1670214	1.297979e+06	1.670214e+06	1.670213e+06	7.743700e+05	
unique	NaN	NaN	4	NaN	NaN	NaN	NaN	
top	NaN	NaN	Cash loans	NaN	NaN	NaN	NaN	
freq	NaN	NaN	747553	NaN	NaN	NaN	NaN	
mean	1.923089e+06	2.783572e+05	NaN	1.595512e+04	1.752339e+05	1.961140e+05	6.697402e+03	
std	5.325980e+05	1.028148e+05	NaN	1.478214e+04	2.927798e+05	3.185746e+05	2.092150e+04	
min	1.000001e+06	1.000010e+05	NaN	0.000000e+00	0.000000e+00	0.000000e+00	-9.000000e-01	
25%	1.461857e+06	1.893290e+05	NaN	6.321780e+03	1.872000e+04	2.416050e+04	0.000000e+00	
50%	1.923110e+06	2.787145e+05	NaN	1.125000e+04	7.104600e+04	8.054100e+04	1.638000e+03	
75%	2.384280e+06	3.675140e+05	NaN	2.065842e+04	1.803600e+05	2.164185e+05	7.740000e+03	
max	2.845382e+06	4.562550e+05	NaN	4.180581e+05	6.905160e+06	6.905160e+06	3.060045e+06	

11 rows × 37 columns

In [25]:
```
datasets["previous_application"].corr()
```

Out[25]:

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AM
SK_ID_PREV	1.000000	-0.000321	0.011459	0.003302	0.003659	-0.001313	
SK_ID_CURR	-0.000321	1.000000	0.000577	0.000280	0.000195	-0.000063	
AMT_ANNUITY	0.011459	0.000577	1.000000	0.808872	0.816429	0.267694	
AMT_APPLICATION	0.003302	0.000280	0.808872	1.000000	0.975824	0.482776	
AMT_CREDIT	0.003659	0.000195	0.816429	0.975824	1.000000	0.301284	
AMT_DOWN_PAYMENT	-0.001313	-0.000063	0.267694	0.482776	0.301284	1.000000	
AMT_GOODS_PRICE	0.015293	0.000369	0.820895	0.999884	0.993087	0.482776	
HOUR_APPR_PROCESS_START	-0.002652	0.002842	-0.036201	-0.014415	-0.021039	0.016776	
NFLAG_LAST_APPL_IN_DAY	-0.002828	0.000098	0.020639	0.004310	-0.025179	0.001597	
RATE_DOWN_PAYMENT	-0.004051	0.001158	-0.103878	-0.072479	-0.188128	0.473935	
RATE_INTEREST_PRIMARY	0.012969	0.033197	0.141823	0.110001	0.125106	0.016323	
RATE_INTEREST_PRIVILEGED	-0.022312	-0.016757	-0.202335	-0.199733	-0.205158	-0.115343	
DAYS_DECISION	0.019100	-0.000637	0.279051	0.133660	0.133763	-0.024536	
SELLERPLACE_AREA	-0.001079	0.001265	-0.015027	-0.007649	-0.009567	0.003533	
CNT_PAYMENT	0.015589	0.000031	0.394535	0.680630	0.674278	0.031659	
DAYS_FIRST_DRAWING	-0.001478	-0.001329	0.052839	0.074544	-0.036813	-0.001773	
DAYS_FIRST_DUE	-0.000071	-0.000757	-0.053295	-0.049532	0.002881	-0.013586	
DAYS_LAST_DUE_1ST_VERSION	0.001222	0.000252	-0.068877	-0.084905	0.044031	-0.000869	
DAYS_LAST_DUE	0.001915	-0.000318	0.082659	0.172627	0.224829	-0.031425	
DAYS_TERMINATION	0.001781	-0.000020	0.068022	0.148618	0.214320	-0.030702	
NFLAG_INSURED_ON_APPROVAL	0.003986	0.000876	0.283080	0.259219	0.263932	-0.042585	

21 rows × 21 columns

Missing data for previous_application

In [26]:
```
percent = (datasets["previous_application"].isnull().sum()/datasets["previous_application"].isnull().count()*10
sum_missing = datasets["previous_application"].isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Test Missing Coun
missing_application_train_data.head(20)
```

Out[26]:

	Percent	Test Missing Count
RATE_INTEREST_PRIVILEGED	99.64	1664263
RATE_INTEREST_PRIMARY	99.64	1664263
AMT_DOWN_PAYMENT	53.64	895844
RATE_DOWN_PAYMENT	53.64	895844
NAME_TYPE_SUITE	49.12	820405
NFLAG_INSURED_ON_APPROVAL	40.30	673065
DAYS_TERMINATION	40.30	673065
DAYS_LAST_DUE	40.30	673065
DAYS_LAST_DUE_1ST_VERSION	40.30	673065
DAYS_FIRST_DUE	40.30	673065
DAYS_FIRST_DRAWING	40.30	673065
AMT_GOODS_PRICE	23.08	385515
AMT_ANNUITY	22.29	372235
CNT_PAYMENT	22.29	372230
PRODUCT_COMBINATION	0.02	346
AMT_CREDIT	0.00	1
NAME_YIELD_GROUP	0.00	0
NAME_PORTFOLIO	0.00	0
NAME_SELLER_INDUSTRY	0.00	0
SELLERPLACE_AREA	0.00	0

In [ ]:
```
plot_missing_data("previous_application",18,20)
```

Summary of installments_payments

In [27]:
```
datasets["installments_payments"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Dtype  
---  ------                  -----  
 0   SK_ID_PREV              int64  
 1   SK_ID_CURR              int64  
 2   NUM_INSTALMENT_VERSION  float64
 3   NUM_INSTALMENT_NUMBER   int64  
 4   DAYS_INSTALMENT         float64
 5   DAYS_ENTRY_PAYMENT      float64
 6   AMT_INSTALMENT          float64
 7   AMT_PAYMENT             float64
dtypes: float64(5), int64(3)
memory usage: 830.4 MB
```

In [29]:
```
datasets["installments_payments"].columns
```

Out[29]:
```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NUM_INSTALMENT_VERSION',
       'NUM_INSTALMENT_NUMBER', 'DAYS_INSTALMENT', 'DAYS_ENTRY_PAYMENT',
       'AMT_INSTALMENT', 'AMT_PAYMENT'],
      dtype='object')
```

In [32]:
```
datasets["installments_payments"].dtypes
```

Out[32]:
```
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
```

In [33]:
```
datasets["installments_payments"].describe()
```

Out[33]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT
count	1.360540e+07	1.360540e+07	1.360540e+07	1.360540e+07	1.360540e+07	1.360250e+07
mean	1.903365e+06	2.784449e+05	8.566373e-01	1.887090e+01	-1.042270e+03	-1.051114e+03
std	5.362029e+05	1.027183e+05	1.035216e+00	2.666407e+01	8.009463e+02	8.005859e+02
min	1.000001e+06	1.000010e+05	0.000000e+00	1.000000e+00	-2.922000e+03	-4.921000e+03
25%	1.434191e+06	1.896390e+05	0.000000e+00	4.000000e+00	-1.654000e+03	-1.662000e+03
50%	1.896520e+06	2.786850e+05	1.000000e+00	8.000000e+00	-8.180000e+02	-8.270000e+02
75%	2.369094e+06	3.675300e+05	1.000000e+00	1.900000e+01	-3.610000e+02	-3.700000e+02
max	2.843499e+06	4.562550e+05	1.780000e+02	2.770000e+02	-1.000000e+00	-1.000000e+00

In [34]:
```
datasets["installments_payments"].describe(include='all')
```

Out[34]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT
count	1.360540e+07	1.360540e+07	1.360540e+07	1.360540e+07	1.360540e+07	1.360250e+07
mean	1.903365e+06	2.784449e+05	8.566373e-01	1.887090e+01	-1.042270e+03	-1.051114e+03
std	5.362029e+05	1.027183e+05	1.035216e+00	2.666407e+01	8.009463e+02	8.005859e+02
min	1.000001e+06	1.000010e+05	0.000000e+00	1.000000e+00	-2.922000e+03	-4.921000e+03
25%	1.434191e+06	1.896390e+05	0.000000e+00	4.000000e+00	-1.654000e+03	-1.662000e+03
50%	1.896520e+06	2.786850e+05	1.000000e+00	8.000000e+00	-8.180000e+02	-8.270000e+02
75%	2.369094e+06	3.675300e+05	1.000000e+00	1.900000e+01	-3.610000e+02	-3.700000e+02
max	2.843499e+06	4.562550e+05	1.780000e+02	2.770000e+02	-1.000000e+00	-1.000000e+00

In [35]:
```
datasets["installments_payments"].corr()
```

Out[35]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALMENT	DA
SK_ID_PREV	1.000000	0.002132	0.000685	-0.002095	0.003748	
SK_ID_CURR	0.002132	1.000000	0.000480	-0.000548	0.001191	
NUM_INSTALMENT_VERSION	0.000685	0.000480	1.000000	-0.323414	0.130244	
NUM_INSTALMENT_NUMBER	-0.002095	-0.000548	-0.323414	1.000000	0.090286	
DAYS_INSTALMENT	0.003748	0.001191	0.130244	0.090286	1.000000	
DAYS_ENTRY_PAYMENT	0.003734	0.001215	0.128124	0.094305	0.999491	
AMT_INSTALMENT	0.002042	-0.000226	0.168109	-0.089640	0.125985	
AMT_PAYMENT	0.001887	-0.000124	0.177176	-0.087664	0.127018	

Missing data for installments_payments

In [36]:
```
percent = (datasets["installments_payments"].isnull().sum()/datasets["installments_payments"].isnull().count()*
sum_missing = datasets["installments_payments"].isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Test Missing Coun
missing_application_train_data.head(20)
```

Out[36]:

	Percent	Test Missing Count
DAYS_ENTRY_PAYMENT	0.02	2905
AMT_PAYMENT	0.02	2905
SK_ID_PREV	0.00	0
SK_ID_CURR	0.00	0
NUM_INSTALMENT_VERSION	0.00	0
NUM_INSTALMENT_NUMBER	0.00	0
DAYS_INSTALMENT	0.00	0
AMT_INSTALMENT	0.00	0

Exploratory Data Analysis

Dataset Size

In [6]:
```
for ds_name in datasets.keys():
    print(f'dataset {ds_name:24}: [ {datasets[ds_name].shape[0]:10,}, {datasets[ds_name].shape[1]}]')
```

```
dataset application_train       : [    307,511, 122]
dataset application_test        : [     48,744, 121]
dataset bureau                  : [  1,716,428, 17]
dataset bureau_balance          : [ 27,299,925, 3]
dataset credit_card_balance     : [  3,840,312, 23]
dataset installments_payments   : [ 13,605,401, 8]
dataset previous_application    : [  1,670,214, 37]
dataset POS_CASH_balance        : [  3,829,580, 8]
```

Function to plot the missing values

In [7]:
```
def plot_missing_data(df, x, y):
    g = sns.displot(
        data=datasets[df].isna().melt(value_name="missing"),
        y="variable",
        hue="missing",
        multiple="fill",
        aspect=1.25
    )
    g.fig.set_figwidth(x)
    g.fig.set_figheight(y)
```

Summary of Application train

In [15]:
```
datasets["application_train"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
```

In [16]:
```
datasets["application_train"].columns
```

Out[16]:
```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY',
       ...
       'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20',
       'FLAG_DOCUMENT_21', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR'],
      dtype='object', length=122)
```

In [17]:
```
datasets["application_train"].dtypes
```

Out[17]:
```
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
```

In [18]:
```
datasets["application_train"].describe() #numerical only features
```

Out[18]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_PO
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.000000	3.072330e+05	
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.573909	5.383962e+05	
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.737315	3.694465e+05	
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.500000	4.050000e+04	
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.000000	2.385000e+05	
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.000000	4.500000e+05	
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05	
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	

8 rows × 106 columns

In [19]:
```
datasets["application_train"].describe(include='all')
```

Out[19]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AI
count	307511.000000	307511.000000	307511	307511	307511	307511	307511.000000	
unique	NaN	NaN	2	3	2	2	NaN	
top	NaN	NaN	Cash loans	F	N	Y	NaN	
freq	NaN	NaN	278232	202448	202924	213312	NaN	
mean	278180.518577	0.080729	NaN	NaN	NaN	NaN	0.417052	
std	102790.175348	0.272419	NaN	NaN	NaN	NaN	0.722121	
min	100002.000000	0.000000	NaN	NaN	NaN	NaN	0.000000	
25%	189145.500000	0.000000	NaN	NaN	NaN	NaN	0.000000	
50%	278202.000000	0.000000	NaN	NaN	NaN	NaN	0.000000	
75%	367142.500000	0.000000	NaN	NaN	NaN	NaN	1.000000	
max	456255.000000	1.000000	NaN	NaN	NaN	NaN	19.000000	

11 rows × 122 columns

In [20]:
```
datasets["application_train"].corr()
```

Out[20]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOO
SK_ID_CURR	1.000000	-0.002108	-0.001129	-0.001820	-0.000343	-0.000433	
TARGET	-0.002108	1.000000	0.019187	-0.003982	-0.030369	-0.012817	
CNT_CHILDREN	-0.001129	0.019187	1.000000	0.012882	0.002145	0.021374	
AMT_INCOME_TOTAL	-0.001820	-0.003982	0.012882	1.000000	0.156870	0.191657	
AMT_CREDIT	-0.000343	-0.030369	0.002145	0.156870	1.000000	0.770138	
...	...	...	...	...	...	...	
AMT_REQ_CREDIT_BUREAU_DAY	-0.002193	0.002704	-0.000366	0.002944	0.004238	0.002185	
AMT_REQ_CREDIT_BUREAU_WEEK	0.002099	0.000788	-0.002436	0.002387	-0.001275	0.013881	
AMT_REQ_CREDIT_BUREAU_MON	0.000485	-0.012462	-0.010808	0.024700	0.054451	0.039148	
AMT_REQ_CREDIT_BUREAU_QRT	0.001025	-0.002022	-0.007836	0.004859	0.015925	0.010124	
AMT_REQ_CREDIT_BUREAU_YEAR	0.004659	0.019930	-0.041550	0.011690	-0.048448	-0.011320	

106 rows × 106 columns

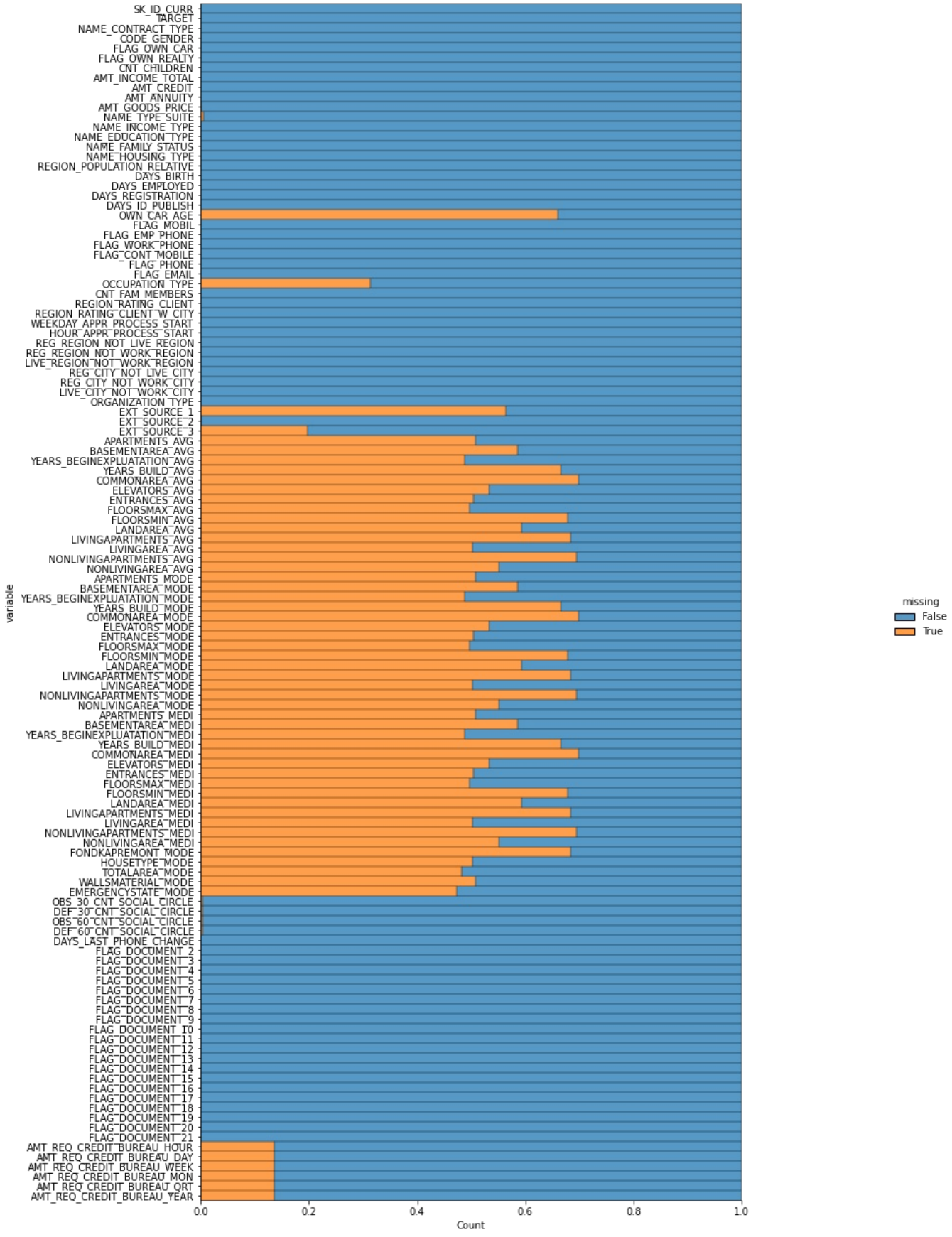
Missing values in Application Train

In [21]:
```
percent = (datasets["application_train"].isnull().sum()/datasets["application_train"].isnull().count()*100).sor
sum_missing = datasets["application_train"].isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Train Missing Cou
missing_application_train_data.head(20)
```

Out[21]:

	Percent	Train Missing Count
COMMONAREA_MEDI	69.87	214865
COMMONAREA_AVG	69.87	214865
COMMONAREA_MODE	69.87	214865
NONLIVINGAPARTMENTS_MODE	69.43	213514
NONLIVINGAPARTMENTS_AVG	69.43	213514
NONLIVINGAPARTMENTS_MEDI	69.43	213514
FONDKAPREMONT_MODE	68.39	210295
LIVINGAPARTMENTS_MODE	68.35	210199
LIVINGAPARTMENTS_AVG	68.35	210199
LIVINGAPARTMENTS_MEDI	68.35	210199
FLOORSMIN_AVG	67.85	208642
FLOORSMIN_MODE	67.85	208642
FLOORSMIN_MEDI	67.85	208642
YEARS_BUILD_MEDI	66.50	204488
YEARS_BUILD_MODE	66.50	204488
YEARS_BUILD_AVG	66.50	204488
OWN_CAR_AGE	65.99	202929
LANDAREA_MEDI	59.38	182590
LANDAREA_MODE	59.38	182590
LANDAREA_AVG	59.38	182590

In [22]:
```
plot_missing_data("application_train",18,20)
```



Summary of Application Test

In [23]:
```
datasets["application_test"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48744 entries, 0 to 48743
Columns: 121 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(40), object(16)
memory usage: 45.0+ MB
```

In [24]:
```
datasets["application_test"].columns
```

Out[24]:
```
Index(['SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR',
       'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT',
       'AMT_ANNUITY', 'AMT_GOODS_PRICE',
       ...
       'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20',
       'FLAG_DOCUMENT_21', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR'],
      dtype='object', length=121)
```

In [25]:
```
datasets["application_test"].dtypes
```

Out[25]:
```
SK_ID_CURR                     int64
NAME_CONTRACT_TYPE            object
CODE_GENDER                   object
FLAG_OWN_CAR                  object
FLAG_OWN_REALTY               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: 121, dtype: object
```

In [26]:
```
datasets["application_test"].describe() #numerical only features
```

Out[26]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_REL
count	48744.000000	48744.000000	4.874400e+04	4.874400e+04	48720.000000	4.874400e+04	48744.0
mean	277796.676350	0.397054	1.784318e+05	5.167404e+05	29426.240209	4.626188e+05	0.0
std	103169.547296	0.709047	1.015226e+05	3.653970e+05	16016.368315	3.367102e+05	0.0
min	100001.000000	0.000000	2.694150e+04	4.500000e+04	2295.000000	4.500000e+04	0.0
25%	188557.750000	0.000000	1.125000e+05	2.606400e+05	17973.000000	2.250000e+05	0.0
50%	277549.000000	0.000000	1.575000e+05	4.500000e+05	26199.000000	3.960000e+05	0.0
75%	367555.500000	1.000000	2.250000e+05	6.750000e+05	37390.500000	6.300000e+05	0.0
max	456250.000000	20.000000	4.410000e+06	2.245500e+06	180576.000000	2.245500e+06	0.0

8 rows × 105 columns

In [27]:
```
datasets["application_test"].describe(include='all') #look at all categorical and numerical
```

Out[27]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TO
count	48744.000000	48744	48744	48744	48744	48744.000000	4.874400e
unique	NaN	2	2	2	2	NaN	
top	NaN	Cash loans	F	N	Y	NaN	
freq	NaN	48305	32678	32311	33658	NaN	
mean	277796.676350	NaN	NaN	NaN	NaN	0.397054	1.784318e
std	103169.547296	NaN	NaN	NaN	NaN	0.709047	1.015226e
min	100001.000000	NaN	NaN	NaN	NaN	0.000000	2.694150e
25%	188557.750000	NaN	NaN	NaN	NaN	0.000000	1.125000e
50%	277549.000000	NaN	NaN	NaN	NaN	0.000000	1.575000e
75%	367555.500000	NaN	NaN	NaN	NaN	1.000000	2.250000e
max	456250.000000	NaN	NaN	NaN	NaN	20.000000	4.410000e

11 rows × 121 columns

In [28]:
```
datasets["application_test"].corr()
```

Out[28]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE
SK_ID_CURR	1.000000	0.000635	0.001278	0.005014	0.007112	0.005097
CNT_CHILDREN	0.000635	1.000000	0.038962	0.027840	0.056770	0.025507
AMT_INCOME_TOTAL	0.001278	0.038962	1.000000	0.396572	0.457833	0.401995
AMT_CREDIT	0.005014	0.027840	0.396572	1.000000	0.777733	0.988056
AMT_ANNUITY	0.007112	0.056770	0.457833	0.777733	1.000000	0.787033
...	...	...	...	...	...	...
AMT_REQ_CREDIT_BUREAU_DAY	0.001083	0.001539	0.004989	0.004882	0.006681	0.004865
AMT_REQ_CREDIT_BUREAU_WEEK	0.001178	0.007523	-0.002867	0.002904	0.003085	0.003358
AMT_REQ_CREDIT_BUREAU_MON	0.000430	-0.008337	0.008691	-0.000156	0.005695	-0.000254
AMT_REQ_CREDIT_BUREAU_QRT	-0.002092	0.029006	0.007410	-0.007750	0.012443	-0.008490
AMT_REQ_CREDIT_BUREAU_YEAR	0.003457	-0.039265	0.003281	-0.034533	-0.044901	-0.036227

105 rows × 105 columns

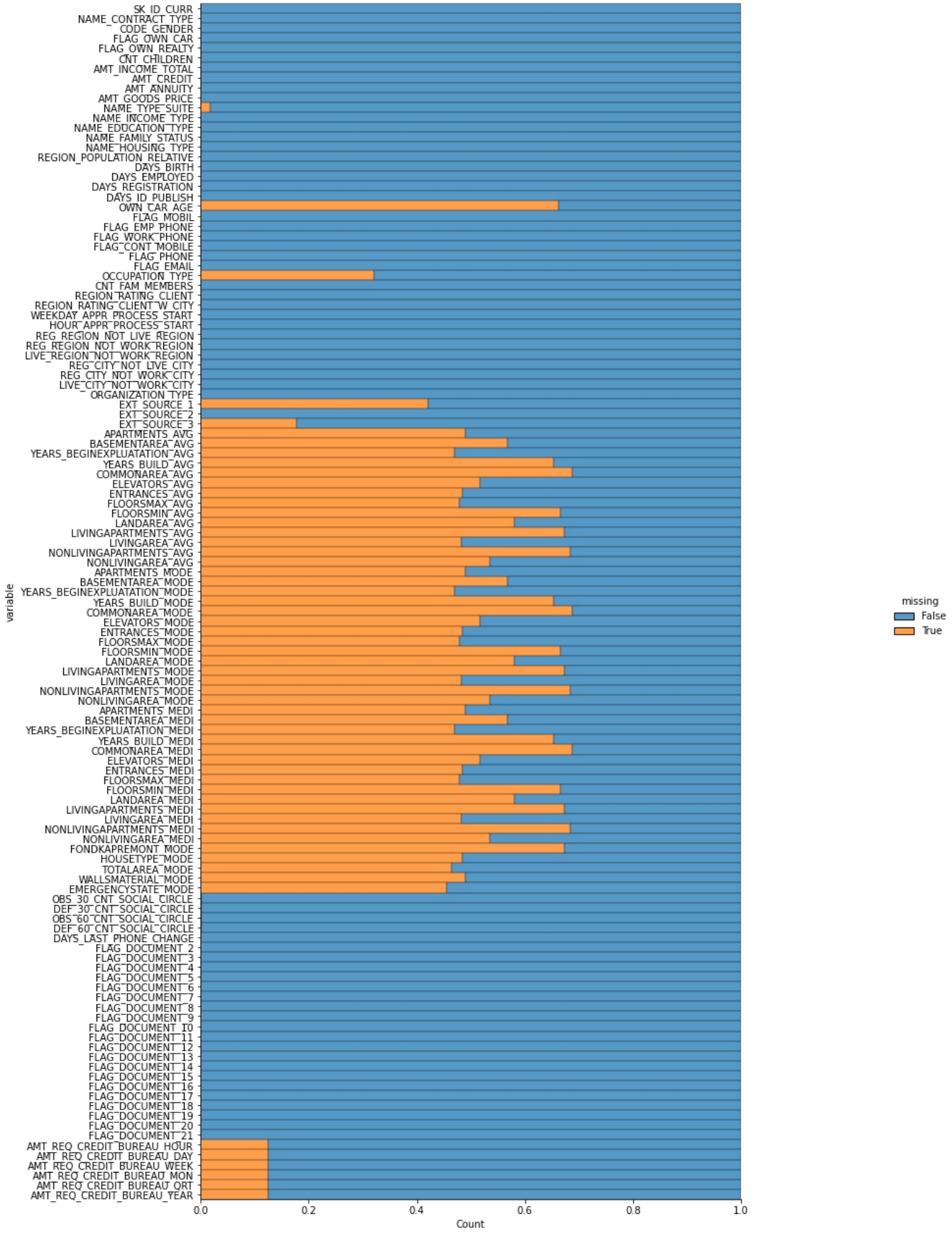
Missing data for Application Test

In [29]:
```
percent = (datasets["application_test"].isnull().sum()/datasets["application_test"].isnull().count()*100).sort_
sum_missing = datasets["application_test"].isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Train Missing Cou
missing_application_train_data.head(20)
```

Out[29]:

	Percent	Train Missing Count
COMMONAREA_AVG	68.72	33495
COMMONAREA_MODE	68.72	33495
COMMONAREA_MEDI	68.72	33495
NONLIVINGAPARTMENTS_AVG	68.41	33347
NONLIVINGAPARTMENTS_MODE	68.41	33347
NONLIVINGAPARTMENTS_MEDI	68.41	33347
FONDKAPREMONT_MODE	67.28	32797
LIVINGAPARTMENTS_AVG	67.25	32780
LIVINGAPARTMENTS_MODE	67.25	32780
LIVINGAPARTMENTS_MEDI	67.25	32780
FLOORSMIN_MEDI	66.61	32466
FLOORSMIN_AVG	66.61	32466
FLOORSMIN_MODE	66.61	32466
OWN_CAR_AGE	66.29	32312
YEARS_BUILD_AVG	65.28	31818
YEARS_BUILD_MEDI	65.28	31818
YEARS_BUILD_MODE	65.28	31818
LANDAREA_MEDI	57.96	28254
LANDAREA_AVG	57.96	28254
LANDAREA_MODE	57.96	28254

In [30]:
```
plot_missing_data("application_test",18,20)
```



Summary of Bureau

In [31]:
```
datasets["bureau"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1716428 entries, 0 to 1716427
Data columns (total 17 columns):
 #   Column                  Dtype  
---  ------                  -----  
 0   SK_ID_CURR              int64  
 1   SK_ID_BUREAU            int64  
 2   CREDIT_ACTIVE           object 
 3   CREDIT_CURRENCY         object 
 4   DAYS_CREDIT             int64  
 5   CREDIT_DAY_OVERDUE      int64  
 6   DAYS_CREDIT_ENDDATE     float64
 7   DAYS_ENDDATE_FACT       float64
 8   AMT_CREDIT_MAX_OVERDUE  float64
 9   CNT_CREDIT_PROLONG      int64  
 10  AMT_CREDIT_SUM          float64
 11  AMT_CREDIT_SUM_DEBT     float64
 12  AMT_CREDIT_SUM_LIMIT    float64
 13  AMT_CREDIT_SUM_OVERDUE  float64
 14  CREDIT_TYPE             object 
 15  DAYS_CREDIT_UPDATE      int64  
 16  AMT_ANNUITY             float64
dtypes: float64(8), int64(6), object(3)
memory usage: 222.6+ MB
```

In [32]:
```
datasets["bureau"].columns
```

Out[32]:
```
Index(['SK_ID_CURR', 'SK_ID_BUREAU', 'CREDIT_ACTIVE', 'CREDIT_CURRENCY',
       'DAYS_CREDIT', 'CREDIT_DAY_OVERDUE', 'DAYS_CREDIT_ENDDATE',
       'DAYS_ENDDATE_FACT', 'AMT_CREDIT_MAX_OVERDUE', 'CNT_CREDIT_PROLONG',
       'AMT_CREDIT_SUM', 'AMT_CREDIT_SUM_DEBT', 'AMT_CREDIT_SUM_LIMIT',
       'AMT_CREDIT_SUM_OVERDUE', 'CREDIT_TYPE', 'DAYS_CREDIT_UPDATE',
       'AMT_ANNUITY'],
      dtype='object')
```

In [33]:
```
datasets["bureau"].dtypes
```

Out[33]:
```
SK_ID_CURR                 int64
SK_ID_BUREAU               int64
CREDIT_ACTIVE             object
CREDIT_CURRENCY           object
DAYS_CREDIT                int64
CREDIT_DAY_OVERDUE         int64
DAYS_CREDIT_ENDDATE      float64
DAYS_ENDDATE_FACT        float64
AMT_CREDIT_MAX_OVERDUE   float64
CNT_CREDIT_PROLONG         int64
AMT_CREDIT_SUM           float64
AMT_CREDIT_SUM_DEBT      float64
AMT_CREDIT_SUM_LIMIT     float64
AMT_CREDIT_SUM_OVERDUE   float64
CREDIT_TYPE               object
DAYS_CREDIT_UPDATE         int64
AMT_ANNUITY              float64
dtype: object
```

In [34]:
```
datasets["bureau"].describe()
```

Out[34]:

	SK_ID_CURR	SK_ID_BUREAU	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	DAYS_ENDDATE_FACT	AMT_CREDI
count	1.716428e+06	1.716428e+06	1.716428e+06	1.716428e+06	1.610875e+06	1.082775e+06	
mean	2.782149e+05	5.924434e+06	-1.142108e+03	8.181666e-01	5.105174e+02	-1.017437e+03	
std	1.029386e+05	5.322657e+05	7.951649e+02	3.654443e+01	4.994220e+03	7.140106e+02	
min	1.000010e+05	5.000000e+06	-2.922000e+03	0.000000e+00	-4.206000e+04	-4.202300e+04	
25%	1.888668e+05	5.463954e+06	-1.666000e+03	0.000000e+00	-1.138000e+03	-1.489000e+03	
50%	2.780550e+05	5.926304e+06	-9.870000e+02	0.000000e+00	-3.300000e+02	-8.970000e+02	
75%	3.674260e+05	6.385681e+06	-4.740000e+02	0.000000e+00	4.740000e+02	-4.250000e+02	
max	4.562550e+05	6.843457e+06	0.000000e+00	2.792000e+03	3.119900e+04	0.000000e+00	

In [35]:
```
datasets["bureau"].describe(include='all')
```

Out[35]:

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDA
count	1.716428e+06	1.716428e+06	1716428	1716428	1.716428e+06	1.716428e+06	1.610875e+
unique	NaN	NaN	4	4	NaN	NaN	N
top	NaN	NaN	Closed	currency 1	NaN	NaN	N
freq	NaN	NaN	1079273	1715020	NaN	NaN	N
mean	2.782149e+05	5.924434e+06	NaN	NaN	-1.142108e+03	8.181666e-01	5.105174e+
std	1.029386e+05	5.322657e+05	NaN	NaN	7.951649e+02	3.654443e+01	4.994220e+
min	1.000010e+05	5.000000e+06	NaN	NaN	-2.922000e+03	0.000000e+00	-4.206000e+
25%	1.888668e+05	5.463954e+06	NaN	NaN	-1.666000e+03	0.000000e+00	-1.138000e+
50%	2.780550e+05	5.926304e+06	NaN	NaN	-9.870000e+02	0.000000e+00	-3.300000e+
75%	3.674260e+05	6.385681e+06	NaN	NaN	-4.740000e+02	0.000000e+00	4.740000e+
max	4.562550e+05	6.843457e+06	NaN	NaN	0.000000e+00	2.792000e+03	3.119900e+

In [36]:
```
datasets["bureau"].corr()
```

Out[36]:

	SK_ID_CURR	SK_ID_BUREAU	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	DAYS_ENDD
SK_ID_CURR	1.000000	0.000135	0.000266	0.000283	0.000456	
SK_ID_BUREAU	0.000135	1.000000	0.013015	-0.002628	0.009107	
DAYS_CREDIT	0.000266	0.013015	1.000000	-0.027266	0.225682	
CREDIT_DAY_OVERDUE	0.000283	-0.002628	-0.027266	1.000000	-0.007352	
DAYS_CREDIT_ENDDATE	0.000456	0.009107	0.225682	-0.007352	1.000000	
DAYS_ENDDATE_FACT	-0.000648	0.017890	0.875359	-0.008637	0.248825	
AMT_CREDIT_MAX_OVERDUE	0.001329	0.002290	-0.014724	0.001249	0.000577	
CNT_CREDIT_PROLONG	-0.000388	-0.000740	-0.030460	0.002756	0.113683	
AMT_CREDIT_SUM	0.001179	0.007962	0.050883	-0.003292	0.055424	
AMT_CREDIT_SUM_DEBT	-0.000790	0.005732	0.135397	-0.002355	0.081298	
AMT_CREDIT_SUM_LIMIT	-0.000304	-0.003986	0.025140	-0.000345	0.095421	
AMT_CREDIT_SUM_OVERDUE	-0.000014	-0.000499	-0.000383	0.090951	0.001077	
DAYS_CREDIT_UPDATE	0.000510	0.019398	0.688771	-0.018461	0.248525	
AMT_ANNUITY	-0.002727	0.001799	0.005676	-0.000339	0.000475	

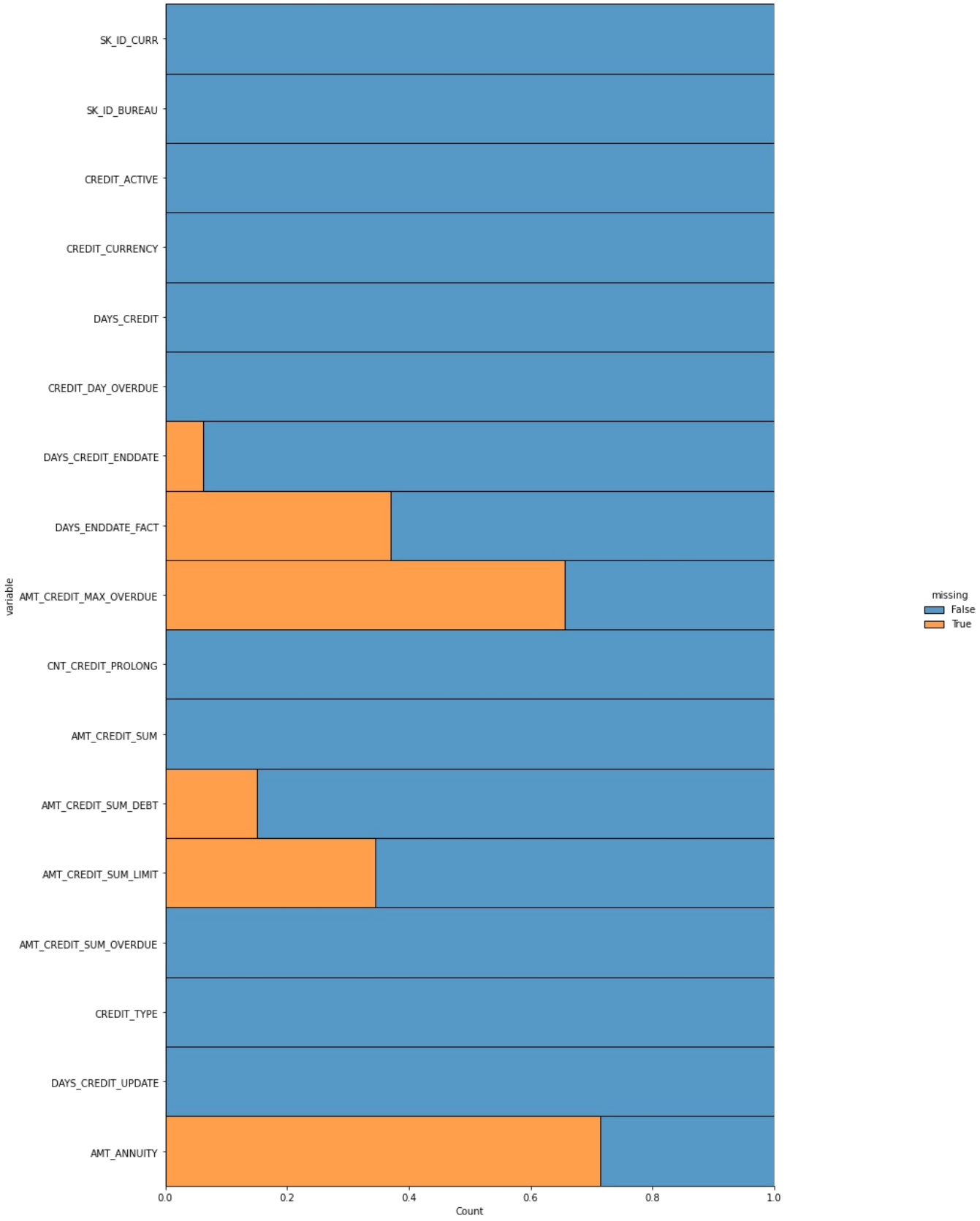
Missing data for Bureau

In [37]:
```
percent = (datasets["bureau"].isnull().sum()/datasets["bureau"].isnull().count()*100).sort_values(ascending = F
sum_missing = datasets["bureau"].isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Test Missing Coun
missing_application_train_data.head(20)
```

Out[37]:

	Percent	Test Missing Count
AMT_ANNUITY	71.47	1226791
AMT_CREDIT_MAX_OVERDUE	65.51	1124488
DAYS_ENDDATE_FACT	36.92	633653
AMT_CREDIT_SUM_LIMIT	34.48	591780
AMT_CREDIT_SUM_DEBT	15.01	257669
DAYS_CREDIT_ENDDATE	6.15	105553
AMT_CREDIT_SUM	0.00	13
CREDIT_ACTIVE	0.00	0
CREDIT_CURRENCY	0.00	0
DAYS_CREDIT	0.00	0
CREDIT_DAY_OVERDUE	0.00	0
SK_ID_BUREAU	0.00	0
CNT_CREDIT_PROLONG	0.00	0
AMT_CREDIT_SUM_OVERDUE	0.00	0
CREDIT_TYPE	0.00	0
DAYS_CREDIT_UPDATE	0.00	0
SK_ID_CURR	0.00	0

In [38]:
```
plot_missing_data("bureau",18,20)
```



Summary of Bureau Balance

In [8]:
```
datasets["bureau_balance"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27299925 entries, 0 to 27299924
Data columns (total 3 columns):
 #   Column          Dtype
---  ------          -----
 0   SK_ID_BUREAU    int64
 1   MONTHS_BALANCE  int64
 2   STATUS          object
dtypes: int64(2), object(1)
memory usage: 624.8+ MB
```

In [9]:
```
datasets["bureau_balance"].columns
```

Out[9]:
```
Index(['SK_ID_BUREAU', 'MONTHS_BALANCE', 'STATUS'], dtype='object')
```

In [10]:
```
datasets["bureau_balance"].dtypes
```

Out[10]:
```
SK_ID_BUREAU       int64
MONTHS_BALANCE     int64
STATUS            object
dtype: object
```

In [11]:
```
datasets["bureau_balance"].describe()
```

Out[11]:

	SK_ID_BUREAU	MONTHS_BALANCE
count	2.729992e+07	2.729992e+07
mean	6.036297e+06	-3.074169e+01
std	4.923489e+05	2.386451e+01
min	5.001709e+06	-9.600000e+01
25%	5.730933e+06	-4.600000e+01
50%	6.070821e+06	-2.500000e+01
75%	6.431951e+06	-1.100000e+01
max	6.842888e+06	0.000000e+00

In [12]:
```
datasets["bureau_balance"].describe(include='all')
```

Out[12]:

	SK_ID_BUREAU	MONTHS_BALANCE	STATUS
count	2.729992e+07	2.729992e+07	27299925
unique	NaN	NaN	8
top	NaN	NaN	C
freq	NaN	NaN	13646993
mean	6.036297e+06	-3.074169e+01	NaN
std	4.923489e+05	2.386451e+01	NaN
min	5.001709e+06	-9.600000e+01	NaN
25%	5.730933e+06	-4.600000e+01	NaN
50%	6.070821e+06	-2.500000e+01	NaN
75%	6.431951e+06	-1.100000e+01	NaN
max	6.842888e+06	0.000000e+00	NaN

In [13]:
```
datasets["bureau_balance"].corr()
```

Out[13]:

	SK_ID_BUREAU	MONTHS_BALANCE
SK_ID_BUREAU	1.000000	0.011873
MONTHS_BALANCE	0.011873	1.000000

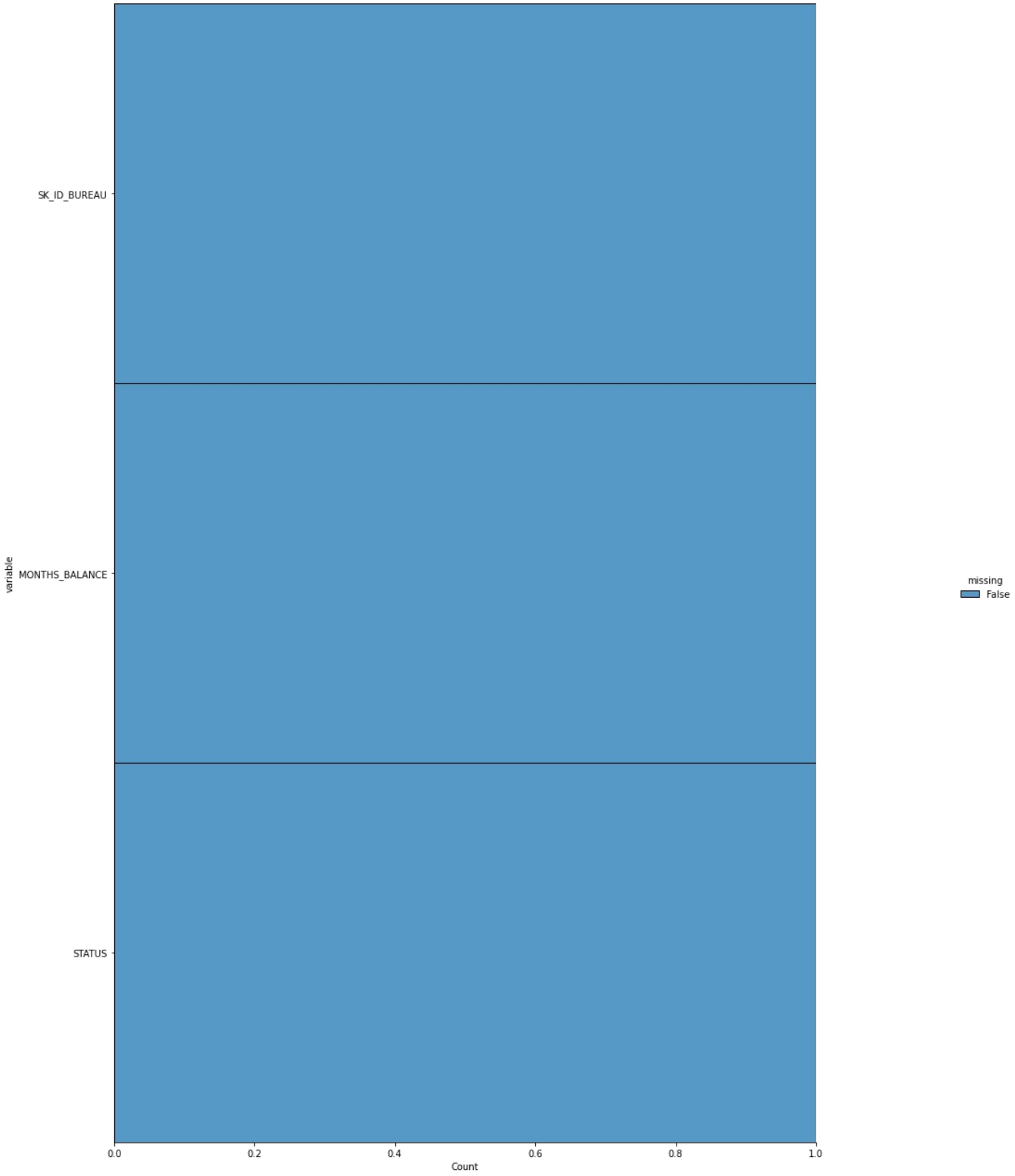
Missing data for Bureau Balance

In [14]:
```
percent = (datasets["bureau_balance"].isnull().sum()/datasets["bureau_balance"].isnull().count()*100).sort_valu
sum_missing = datasets["bureau_balance"].isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Test Missing Coun
missing_application_train_data.head(20)
```

Out[14]:

	Percent	Test Missing Count
SK_ID_BUREAU	0.0	0
MONTHS_BALANCE	0.0	0
STATUS	0.0	0

In [15]:
```
plot_missing_data("bureau_balance",18,20)
```



Summary of POS_CASH_balance

In [6]:
```
datasets["POS_CASH_balance"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3829580 entries, 0 to 3829579
Data columns (total 8 columns):
 #   Column                 Dtype
---  ------                 -----
 0   SK_ID_PREV             int64
 1   SK_ID_CURR             int64
 2   MONTHS_BALANCE         int64
 3   CNT_INSTALMENT         float64
 4   CNT_INSTALMENT_FUTURE  float64
 5   NAME_CONTRACT_STATUS   object
 6   SK_DPD                 float64
 7   SK_DPD_DEF             float64
dtypes: float64(4), int64(3), object(1)
memory usage: 233.7+ MB
```

In [7]:
```
datasets["POS_CASH_balance"].columns
```

Out[7]:
```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'MONTHS_BALANCE', 'CNT_INSTALMENT',
       'CNT_INSTALMENT_FUTURE', 'NAME_CONTRACT_STATUS', 'SK_DPD',
       'SK_DPD_DEF'],
      dtype='object')
```

In [8]:
```
datasets["POS_CASH_balance"].dtypes
```

Out[8]:
```
SK_ID_PREV                int64
SK_ID_CURR                int64
MONTHS_BALANCE            int64
CNT_INSTALMENT          float64
CNT_INSTALMENT_FUTURE   float64
NAME_CONTRACT_STATUS     object
SK_DPD                  float64
SK_DPD_DEF              float64
dtype: object
```

In [9]:
```
datasets["POS_CASH_balance"].describe()
```

Out[9]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FUTURE	SK_DPD	SK_DPD_DEF
count	3.829580e+06	3.829580e+06	3.829580e+06	3.823444e+06	3.823437e+06	3.829579e+06	3.829579e+06
mean	1.904375e+06	2.785338e+05	-3.214404e+01	1.956578e+01	1.283459e+01	4.358176e-01	7.258109e-02
std	5.355338e+05	1.027329e+05	2.549135e+01	1.380046e+01	1.273046e+01	1.744642e+01	1.541065e+00
min	1.000001e+06	1.000010e+05	-9.600000e+01	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	1.435030e+06	1.896800e+05	-4.600000e+01	1.000000e+01	4.000000e+00	0.000000e+00	0.000000e+00
50%	1.898227e+06	2.788660e+05	-2.300000e+01	1.200000e+01	9.000000e+00	0.000000e+00	0.000000e+00
75%	2.369573e+06	3.676380e+05	-1.200000e+01	2.400000e+01	1.800000e+01	0.000000e+00	0.000000e+00
max	2.843499e+06	4.562550e+05	-1.000000e+00	9.200000e+01	8.500000e+01	3.006000e+03	4.190000e+02

In [10]:
```
datasets["POS_CASH_balance"].describe(include='all')
```

Out[10]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FUTURE	NAME_CONTRACT_STATUS	SI
count	3.829580e+06	3.829580e+06	3.829580e+06	3.823444e+06	3.823437e+06	3829579	3.8295
unique	NaN	NaN	NaN	NaN	NaN	8	
top	NaN	NaN	NaN	NaN	NaN	Active	
freq	NaN	NaN	NaN	NaN	NaN	3570142	
mean	1.904375e+06	2.785338e+05	-3.214404e+01	1.956578e+01	1.283459e+01	NaN	4.3581
std	5.355338e+05	1.027329e+05	2.549135e+01	1.380046e+01	1.273046e+01	NaN	1.7446
min	1.000001e+06	1.000010e+05	-9.600000e+01	1.000000e+00	0.000000e+00	NaN	0.0000
25%	1.435030e+06	1.896800e+05	-4.600000e+01	1.000000e+01	4.000000e+00	NaN	0.0000
50%	1.898227e+06	2.788660e+05	-2.300000e+01	1.200000e+01	9.000000e+00	NaN	0.0000
75%	2.369573e+06	3.676380e+05	-1.200000e+01	2.400000e+01	1.800000e+01	NaN	0.0000
max	2.843499e+06	4.562550e+05	-1.000000e+00	9.200000e+01	8.500000e+01	NaN	3.0060

In [11]:
```
datasets["POS_CASH_balance"].corr()
```

Out[11]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FUTURE	SK_DPD	SK_D
SK_ID_PREV	1.000000	-0.000208	0.003497	0.003542	0.003431	0.000632	
SK_ID_CURR	-0.000208	1.000000	0.000430	0.000618	-0.000105	-0.000401	
MONTHS_BALANCE	0.003497	0.000430	1.000000	0.433006	0.351605	-0.010548	
CNT_INSTALMENT	0.003542	0.000618	0.433006	1.000000	0.897199	-0.013366	
CNT_INSTALMENT_FUTURE	0.003431	-0.000105	0.351605	0.897199	1.000000	-0.020738	
SK_DPD	0.000632	-0.000401	-0.010548	-0.013366	-0.020738	1.000000	
SK_DPD_DEF	0.000186	0.002109	-0.027817	-0.009263	-0.017952	0.090650	

Missing data for POS_CASH_balance

In [12]:
```
percent = (datasets["POS_CASH_balance"].isnull().sum()/datasets["POS_CASH_balance"].isnull().count()*100).sort_
sum_missing = datasets["POS_CASH_balance"].isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Test Missing Coun
missing_application_train_data.head(20)
```

Out[12]:

	Percent	Test Missing Count
CNT_INSTALMENT_FUTURE	0.16	6143
CNT_INSTALMENT	0.16	6136
NAME_CONTRACT_STATUS	0.00	1
SK_DPD	0.00	1
SK_DPD_DEF	0.00	1
SK_ID_PREV	0.00	0
SK_ID_CURR	0.00	0
MONTHS_BALANCE	0.00	0

In [ ]:
```
plot_missing_data("POS_CASH_balance",18,20)
```

Summary of credit_card_balance

In [13]:
```
datasets["credit_card_balance"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3840312 entries, 0 to 3840311
Data columns (total 23 columns):
 #   Column                      Dtype  
---  ------                      -----  
 0   SK_ID_PREV                  int64  
 1   SK_ID_CURR                  int64  
 2   MONTHS_BALANCE              int64  
 3   AMT_BALANCE                 float64
 4   AMT_CREDIT_LIMIT_ACTUAL     int64  
 5   AMT_DRAWINGS_ATM_CURRENT    float64
 6   AMT_DRAWINGS_CURRENT        float64
 7   AMT_DRAWINGS_OTHER_CURRENT  float64
 8   AMT_DRAWINGS_POS_CURRENT    float64
 9   AMT_INST_MIN_REGULARITY     float64
 10  AMT_PAYMENT_CURRENT         float64
 11  AMT_PAYMENT_TOTAL_CURRENT   float64
 12  AMT_RECEIVABLE_PRINCIPAL    float64
 13  AMT_RECIVABLE               float64
 14  AMT_TOTAL_RECEIVABLE        float64
 15  CNT_DRAWINGS_ATM_CURRENT    float64
 16  CNT_DRAWINGS_CURRENT        int64  
 17  CNT_DRAWINGS_OTHER_CURRENT  float64
 18  CNT_DRAWINGS_POS_CURRENT    float64
 19  CNT_INSTALMENT_MATURE_CUM   float64
 20  NAME_CONTRACT_STATUS        object 
 21  SK_DPD                      int64  
 22  SK_DPD_DEF                  int64  
dtypes: float64(15), int64(7), object(1)
memory usage: 673.9+ MB
```

In [14]:
```
datasets["credit_card_balance"].columns
```

Out[14]:
```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'MONTHS_BALANCE', 'AMT_BALANCE',
       'AMT_CREDIT_LIMIT_ACTUAL', 'AMT_DRAWINGS_ATM_CURRENT',
       'AMT_DRAWINGS_CURRENT', 'AMT_DRAWINGS_OTHER_CURRENT',
       'AMT_DRAWINGS_POS_CURRENT', 'AMT_INST_MIN_REGULARITY',
       'AMT_PAYMENT_CURRENT', 'AMT_PAYMENT_TOTAL_CURRENT',
       'AMT_RECEIVABLE_PRINCIPAL', 'AMT_RECIVABLE', 'AMT_TOTAL_RECEIVABLE',
       'CNT_DRAWINGS_ATM_CURRENT', 'CNT_DRAWINGS_CURRENT',
       'CNT_DRAWINGS_OTHER_CURRENT', 'CNT_DRAWINGS_POS_CURRENT',
       'CNT_INSTALMENT_MATURE_CUM', 'NAME_CONTRACT_STATUS', 'SK_DPD',
       'SK_DPD_DEF'],
      dtype='object')
```

In [15]:
```
datasets["credit_card_balance"].dtypes
```

Out[15]:
```
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
SK_DPD                       int64
SK_DPD_DEF                   int64
dtype: object
```

In [16]:
```
datasets["credit_card_balance"].describe()
```

Out[16]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_DRAWINGS_ATM_CURRENT	AMT
count	3.840312e+06	3.840312e+06	3.840312e+06	3.840312e+06	3.840312e+06	3.090496e+06	
mean	1.904504e+06	2.783242e+05	-3.452192e+01	5.830016e+04	1.538080e+05	5.961325e+03	
std	5.364695e+05	1.027045e+05	2.666775e+01	1.063070e+05	1.651457e+05	2.822569e+04	
min	1.000018e+06	1.000060e+05	-9.600000e+01	-4.202502e+05	0.000000e+00	-6.827310e+03	
25%	1.434385e+06	1.895170e+05	-5.500000e+01	0.000000e+00	4.500000e+04	0.000000e+00	
50%	1.897122e+06	2.783960e+05	-2.800000e+01	0.000000e+00	1.125000e+05	0.000000e+00	
75%	2.369328e+06	3.675800e+05	-1.100000e+01	8.904669e+04	1.800000e+05	0.000000e+00	
max	2.843496e+06	4.562500e+05	-1.000000e+00	1.505902e+06	1.350000e+06	2.115000e+06	

8 rows × 22 columns

In [17]:
```
datasets["credit_card_balance"].describe(include='all')
```

Out[17]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_DRAWINGS_ATM_CURRENT	AM
count	3.840312e+06	3.840312e+06	3.840312e+06	3.840312e+06	3.840312e+06	3.090496e+06	
unique	NaN	NaN	NaN	NaN	NaN	NaN	
top	NaN	NaN	NaN	NaN	NaN	NaN	
freq	NaN	NaN	NaN	NaN	NaN	NaN	
mean	1.904504e+06	2.783242e+05	-3.452192e+01	5.830016e+04	1.538080e+05	5.961325e+03	
std	5.364695e+05	1.027045e+05	2.666775e+01	1.063070e+05	1.651457e+05	2.822569e+04	
min	1.000018e+06	1.000060e+05	-9.600000e+01	-4.202502e+05	0.000000e+00	-6.827310e+03	
25%	1.434385e+06	1.895170e+05	-5.500000e+01	0.000000e+00	4.500000e+04	0.000000e+00	
50%	1.897122e+06	2.783960e+05	-2.800000e+01	0.000000e+00	1.125000e+05	0.000000e+00	
75%	2.369328e+06	3.675800e+05	-1.100000e+01	8.904669e+04	1.800000e+05	0.000000e+00	
max	2.843496e+06	4.562500e+05	-1.000000e+00	1.505902e+06	1.350000e+06	2.115000e+06	

11 rows × 23 columns

In [18]:
```
datasets["credit_card_balance"].corr()
```

Out[18]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_DRA
SK_ID_PREV	1.000000	0.004723	0.003670	0.005046	0.006631	
SK_ID_CURR	0.004723	1.000000	0.001696	0.003510	0.005991	
MONTHS_BALANCE	0.003670	0.001696	1.000000	0.014558	0.199900	
AMT_BALANCE	0.005046	0.003510	0.014558	1.000000	0.489386	
AMT_CREDIT_LIMIT_ACTUAL	0.006631	0.005991	0.199900	0.489386	1.000000	
AMT_DRAWINGS_ATM_CURRENT	0.004342	0.000814	0.036802	0.283551	0.247219	
AMT_DRAWINGS_CURRENT	0.002624	0.000708	0.065527	0.336965	0.263093	
AMT_DRAWINGS_OTHER_CURRENT	-0.000160	0.000958	0.000405	0.065366	0.050579	
AMT_DRAWINGS_POS_CURRENT	0.001721	-0.000786	0.118146	0.169449	0.234976	
AMT_INST_MIN_REGULARITY	0.006460	0.003300	-0.087529	0.896728	0.467620	
AMT_PAYMENT_CURRENT	0.003472	0.000127	0.076355	0.143934	0.308294	
AMT_PAYMENT_TOTAL_CURRENT	0.001641	0.000784	0.035614	0.151349	0.226570	
AMT_RECEIVABLE_PRINCIPAL	0.005140	0.003589	0.016266	0.999720	0.490445	
AMT_RECIVABLE	0.005035	0.003518	0.013172	0.999917	0.488641	
AMT_TOTAL_RECEIVABLE	0.005032	0.003524	0.013084	0.999897	0.488598	
CNT_DRAWINGS_ATM_CURRENT	0.002821	0.002082	0.002536	0.309968	0.221808	
CNT_DRAWINGS_CURRENT	0.000367	0.002654	0.113321	0.259184	0.204237	
CNT_DRAWINGS_OTHER_CURRENT	-0.001412	-0.000131	-0.026192	0.046563	0.030051	
CNT_DRAWINGS_POS_CURRENT	0.000809	0.002135	0.160207	0.155553	0.202868	
CNT_INSTALMENT_MATURE_CUM	-0.007219	-0.000581	-0.008620	0.005009	-0.157269	
SK_DPD	-0.001786	-0.000962	0.039434	-0.046988	-0.038791	
SK_DPD_DEF	0.001973	0.001519	0.001659	0.013009	-0.002236	

22 rows × 22 columns

Missing data for credit_card_balance

In [19]:
```
percent = (datasets["credit_card_balance"].isnull().sum()/datasets["credit_card_balance"].isnull().count()*100)
sum_missing = datasets["credit_card_balance"].isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Test Missing Coun
missing_application_train_data.head(20)
```

Out[19]:

	Percent	Test Missing Count
AMT_PAYMENT_CURRENT	20.00	767988
AMT_DRAWINGS_ATM_CURRENT	19.52	749816
CNT_DRAWINGS_POS_CURRENT	19.52	749816
AMT_DRAWINGS_OTHER_CURRENT	19.52	749816
AMT_DRAWINGS_POS_CURRENT	19.52	749816
CNT_DRAWINGS_OTHER_CURRENT	19.52	749816
CNT_DRAWINGS_ATM_CURRENT	19.52	749816
CNT_INSTALMENT_MATURE_CUM	7.95	305236
AMT_INST_MIN_REGULARITY	7.95	305236
SK_ID_PREV	0.00	0
AMT_TOTAL_RECEIVABLE	0.00	0
SK_DPD	0.00	0
NAME_CONTRACT_STATUS	0.00	0
CNT_DRAWINGS_CURRENT	0.00	0
AMT_PAYMENT_TOTAL_CURRENT	0.00	0
AMT_RECIVABLE	0.00	0
AMT_RECEIVABLE_PRINCIPAL	0.00	0
SK_ID_CURR	0.00	0
AMT_DRAWINGS_CURRENT	0.00	0
AMT_CREDIT_LIMIT_ACTUAL	0.00	0

In [ ]:
```
plot_missing_data("credit_card_balance",18,20)
```

Summary of previous_application

In [20]:
```
datasets["previous_application"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 37 columns):
 #   Column                       Non-Null Count    Dtype  
---  ------                       --------------    -----  
 0   SK_ID_PREV                   1670214 non-null  int64  
 1   SK_ID_CURR                   1670214 non-null  int64  
 2   NAME_CONTRACT_TYPE           1670214 non-null  object 
 3   AMT_ANNUITY                  1297979 non-null  float64
 4   AMT_APPLICATION              1670214 non-null  float64
 5   AMT_CREDIT                   1670213 non-null  float64
 6   AMT_DOWN_PAYMENT             774370 non-null   float64
 7   AMT_GOODS_PRICE              1284699 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object 
 9   HOUR_APPR_PROCESS_START      1670214 non-null  int64  
 10  FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object 
 11  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64  
 12  RATE_DOWN_PAYMENT            774370 non-null   float64
 13  RATE_INTEREST_PRIMARY        5951 non-null     float64
 14  RATE_INTEREST_PRIVILEGED     5951 non-null     float64
 15  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object 
 16  NAME_CONTRACT_STATUS         1670214 non-null  object 
 17  DAYS_DECISION                1670214 non-null  int64  
 18  NAME_PAYMENT_TYPE            1670214 non-null  object 
 19  CODE_REJECT_REASON           1670214 non-null  object 
 20  NAME_TYPE_SUITE              849809 non-null   object 
 21  NAME_CLIENT_TYPE             1670214 non-null  object 
 22  NAME_GOODS_CATEGORY          1670214 non-null  object 
 23  NAME_PORTFOLIO               1670214 non-null  object 
 24  NAME_PRODUCT_TYPE            1670214 non-null  object 
 25  CHANNEL_TYPE                 1670214 non-null  object 
 26  SELLERPLACE_AREA             1670214 non-null  int64  
 27  NAME_SELLER_INDUSTRY         1670214 non-null  object 
 28  CNT_PAYMENT                  1297984 non-null  float64
 29  NAME_YIELD_GROUP             1670214 non-null  object 
 30  PRODUCT_COMBINATION          1669868 non-null  object 
 31  DAYS_FIRST_DRAWING           997149 non-null   float64
 32  DAYS_FIRST_DUE               997149 non-null   float64
 33  DAYS_LAST_DUE_1ST_VERSION    997149 non-null   float64
 34  DAYS_LAST_DUE                997149 non-null   float64
 35  DAYS_TERMINATION             997149 non-null   float64
 36  NFLAG_INSURED_ON_APPROVAL    997149 non-null   float64
dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
```

In [21]:
```
datasets["previous_application"].columns
```

Out[21]:
```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY',
       'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRICE',
       'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START',
       'FLAG_LAST_APPL_PER_CONTRACT', 'NFLAG_LAST_APPL_IN_DAY',
       'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY',
       'RATE_INTEREST_PRIVILEGED', 'NAME_CASH_LOAN_PURPOSE',
       'NAME_CONTRACT_STATUS', 'DAYS_DECISION', 'NAME_PAYMENT_TYPE',
       'CODE_REJECT_REASON', 'NAME_TYPE_SUITE', 'NAME_CLIENT_TYPE',
       'NAME_GOODS_CATEGORY', 'NAME_PORTFOLIO', 'NAME_PRODUCT_TYPE',
       'CHANNEL_TYPE', 'SELLERPLACE_AREA', 'NAME_SELLER_INDUSTRY',
       'CNT_PAYMENT', 'NAME_YIELD_GROUP', 'PRODUCT_COMBINATION',
       'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION',
       'DAYS_LAST_DUE', 'DAYS_TERMINATION', 'NFLAG_INSURED_ON_APPROVAL'],
      dtype='object')
```

In [22]:
```
datasets["previous_application"].dtypes
```

Out[22]:
```
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
NAME_CLIENT_TYPE             object
NAME_GOODS_CATEGORY          object
NAME_PORTFOLIO               object
NAME_PRODUCT_TYPE            object
CHANNEL_TYPE                 object
SELLERPLACE_AREA              int64
NAME_SELLER_INDUSTRY         object
CNT_PAYMENT                 float64
NAME_YIELD_GROUP             object
PRODUCT_COMBINATION          object
DAYS_FIRST_DRAWING          float64
DAYS_FIRST_DUE              float64
DAYS_LAST_DUE_1ST_VERSION   float64
DAYS_LAST_DUE               float64
DAYS_TERMINATION            float64
NFLAG_INSURED_ON_APPROVAL   float64
dtype: object
```

In [23]:
```
datasets["previous_application"].describe()
```

Out[23]:

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	HOUR_AI
count	1.670214e+06	1.670214e+06	1.297979e+06	1.670214e+06	1.670213e+06	7.743700e+05	1.284699e+06	
mean	1.923089e+06	2.783572e+05	1.595512e+04	1.752339e+05	1.961140e+05	6.697402e+03	2.278473e+05	
std	5.325980e+05	1.028148e+05	1.478214e+04	2.927798e+05	3.185746e+05	2.092150e+04	3.153966e+05	
min	1.000001e+06	1.000010e+05	0.000000e+00	0.000000e+00	0.000000e+00	-9.000000e-01	0.000000e+00	
25%	1.461857e+06	1.893290e+05	6.321780e+03	1.872000e+04	2.416050e+04	0.000000e+00	5.084100e+04	
50%	1.923110e+06	2.787145e+05	1.125000e+04	7.104600e+04	8.054100e+04	1.638000e+03	1.123200e+05	
75%	2.384280e+06	3.675140e+05	2.065842e+04	1.803600e+05	2.164185e+05	7.740000e+03	2.340000e+05	
max	2.845382e+06	4.562550e+05	4.180581e+05	6.905160e+06	6.905160e+06	3.060045e+06	6.905160e+06	

8 rows × 21 columns

In [24]:
```
datasets["previous_application"].describe(include='all')
```

Out[24]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AM
count	1.670214e+06	1.670214e+06	1670214	1.297979e+06	1.670214e+06	1.670213e+06	7.743700e+05	
unique	NaN	NaN	4	NaN	NaN	NaN	NaN	
top	NaN	NaN	Cash loans	NaN	NaN	NaN	NaN	
freq	NaN	NaN	747553	NaN	NaN	NaN	NaN	
mean	1.923089e+06	2.783572e+05	NaN	1.595512e+04	1.752339e+05	1.961140e+05	6.697402e+03	
std	5.325980e+05	1.028148e+05	NaN	1.478214e+04	2.927798e+05	3.185746e+05	2.092150e+04	
min	1.000001e+06	1.000010e+05	NaN	0.000000e+00	0.000000e+00	0.000000e+00	-9.000000e-01	
25%	1.461857e+06	1.893290e+05	NaN	6.321780e+03	1.872000e+04	2.416050e+04	0.000000e+00	
50%	1.923110e+06	2.787145e+05	NaN	1.125000e+04	7.104600e+04	8.054100e+04	1.638000e+03	
75%	2.384280e+06	3.675140e+05	NaN	2.065842e+04	1.803600e+05	2.164185e+05	7.740000e+03	
max	2.845382e+06	4.562550e+05	NaN	4.180581e+05	6.905160e+06	6.905160e+06	3.060045e+06	

11 rows × 37 columns

In [25]:
```
datasets["previous_application"].corr()
```

Out[25]:

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AM
SK_ID_PREV	1.000000	-0.000321	0.011459	0.003302	0.003659	-0.001313	
SK_ID_CURR	-0.000321	1.000000	0.000577	0.000280	0.000195	-0.000063	
AMT_ANNUITY	0.011459	0.000577	1.000000	0.808872	0.816429	0.267694	
AMT_APPLICATION	0.003302	0.000280	0.808872	1.000000	0.975824	0.482776	
AMT_CREDIT	0.003659	0.000195	0.816429	0.975824	1.000000	0.301284	
AMT_DOWN_PAYMENT	-0.001313	-0.000063	0.267694	0.482776	0.301284	1.000000	
AMT_GOODS_PRICE	0.015293	0.000369	0.820895	0.999884	0.993087	0.482776	
HOUR_APPR_PROCESS_START	-0.002652	0.002842	-0.036201	-0.014415	-0.021039	0.016776	
NFLAG_LAST_APPL_IN_DAY	-0.002828	0.000098	0.020639	0.004310	-0.025179	0.001597	
RATE_DOWN_PAYMENT	-0.004051	0.001158	-0.103878	-0.072479	-0.188128	0.473935	
RATE_INTEREST_PRIMARY	0.012969	0.033197	0.141823	0.110001	0.125106	0.016323	
RATE_INTEREST_PRIVILEGED	-0.022312	-0.016757	-0.202335	-0.199733	-0.205158	-0.115343	
DAYS_DECISION	0.019100	-0.000637	0.279051	0.133660	0.133763	-0.024536	
SELLERPLACE_AREA	-0.001079	0.001265	-0.015027	-0.007649	-0.009567	0.003533	
CNT_PAYMENT	0.015589	0.000031	0.394535	0.680630	0.674278	0.031659	
DAYS_FIRST_DRAWING	-0.001478	-0.001329	0.052839	0.074544	-0.036813	-0.001773	
DAYS_FIRST_DUE	-0.000071	-0.000757	-0.053295	-0.049532	0.002881	-0.013586	
DAYS_LAST_DUE_1ST_VERSION	0.001222	0.000252	-0.068877	-0.084905	0.044031	-0.000869	
DAYS_LAST_DUE	0.001915	-0.000318	0.082659	0.172627	0.224829	-0.031425	
DAYS_TERMINATION	0.001781	-0.000020	0.068022	0.148618	0.214320	-0.030702	
NFLAG_INSURED_ON_APPROVAL	0.003986	0.000876	0.283080	0.259219	0.263932	-0.042585	

21 rows × 21 columns

Missing data for previous_application

In [26]:
```
percent = (datasets["previous_application"].isnull().sum()/datasets["previous_application"].isnull().count()*10
sum_missing = datasets["previous_application"].isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Test Missing Coun
missing_application_train_data.head(20)
```

Out[26]:

	Percent	Test Missing Count
RATE_INTEREST_PRIVILEGED	99.64	1664263
RATE_INTEREST_PRIMARY	99.64	1664263
AMT_DOWN_PAYMENT	53.64	895844
RATE_DOWN_PAYMENT	53.64	895844
NAME_TYPE_SUITE	49.12	820405
NFLAG_INSURED_ON_APPROVAL	40.30	673065
DAYS_TERMINATION	40.30	673065
DAYS_LAST_DUE	40.30	673065
DAYS_LAST_DUE_1ST_VERSION	40.30	673065
DAYS_FIRST_DUE	40.30	673065
DAYS_FIRST_DRAWING	40.30	673065
AMT_GOODS_PRICE	23.08	385515
AMT_ANNUITY	22.29	372235
CNT_PAYMENT	22.29	372230
PRODUCT_COMBINATION	0.02	346
AMT_CREDIT	0.00	1
NAME_YIELD_GROUP	0.00	0
NAME_PORTFOLIO	0.00	0
NAME_SELLER_INDUSTRY	0.00	0
SELLERPLACE_AREA	0.00	0

In [ ]:
```
plot_missing_data("previous_application",18,20)
```

Summary of installments_payments

In [27]:
```
datasets["installments_payments"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Dtype  
---  ------                  -----  
 0   SK_ID_PREV              int64  
 1   SK_ID_CURR              int64  
 2   NUM_INSTALMENT_VERSION  float64
 3   NUM_INSTALMENT_NUMBER   int64  
 4   DAYS_INSTALMENT         float64
 5   DAYS_ENTRY_PAYMENT      float64
 6   AMT_INSTALMENT          float64
 7   AMT_PAYMENT             float64
dtypes: float64(5), int64(3)
memory usage: 830.4 MB
```

In [29]:
```
datasets["installments_payments"].columns
```

Out[29]:
```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NUM_INSTALMENT_VERSION',
       'NUM_INSTALMENT_NUMBER', 'DAYS_INSTALMENT', 'DAYS_ENTRY_PAYMENT',
       'AMT_INSTALMENT', 'AMT_PAYMENT'],
      dtype='object')
```

In [32]:
```
datasets["installments_payments"].dtypes
```

Out[32]:
```
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
```

In [33]:
```
datasets["installments_payments"].describe()
```

Out[33]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT
count	1.360540e+07	1.360540e+07	1.360540e+07	1.360540e+07	1.360540e+07	1.360250e+07
mean	1.903365e+06	2.784449e+05	8.566373e-01	1.887090e+01	-1.042270e+03	-1.051114e+03
std	5.362029e+05	1.027183e+05	1.035216e+00	2.666407e+01	8.009463e+02	8.005859e+02
min	1.000001e+06	1.000010e+05	0.000000e+00	1.000000e+00	-2.922000e+03	-4.921000e+03
25%	1.434191e+06	1.896390e+05	0.000000e+00	4.000000e+00	-1.654000e+03	-1.662000e+03
50%	1.896520e+06	2.786850e+05	1.000000e+00	8.000000e+00	-8.180000e+02	-8.270000e+02
75%	2.369094e+06	3.675300e+05	1.000000e+00	1.900000e+01	-3.610000e+02	-3.700000e+02
max	2.843499e+06	4.562550e+05	1.780000e+02	2.770000e+02	-1.000000e+00	-1.000000e+00

In [34]:
```
datasets["installments_payments"].describe(include='all')
```

Out[34]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT
count	1.360540e+07	1.360540e+07	1.360540e+07	1.360540e+07	1.360540e+07	1.360250e+07
mean	1.903365e+06	2.784449e+05	8.566373e-01	1.887090e+01	-1.042270e+03	-1.051114e+03
std	5.362029e+05	1.027183e+05	1.035216e+00	2.666407e+01	8.009463e+02	8.005859e+02
min	1.000001e+06	1.000010e+05	0.000000e+00	1.000000e+00	-2.922000e+03	-4.921000e+03
25%	1.434191e+06	1.896390e+05	0.000000e+00	4.000000e+00	-1.654000e+03	-1.662000e+03
50%	1.896520e+06	2.786850e+05	1.000000e+00	8.000000e+00	-8.180000e+02	-8.270000e+02
75%	2.369094e+06	3.675300e+05	1.000000e+00	1.900000e+01	-3.610000e+02	-3.700000e+02
max	2.843499e+06	4.562550e+05	1.780000e+02	2.770000e+02	-1.000000e+00	-1.000000e+00

In [35]:
```
datasets["installments_payments"].corr()
```

Out[35]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALMENT	DA
SK_ID_PREV	1.000000	0.002132	0.000685	-0.002095	0.003748	
SK_ID_CURR	0.002132	1.000000	0.000480	-0.000548	0.001191	
NUM_INSTALMENT_VERSION	0.000685	0.000480	1.000000	-0.323414	0.130244	
NUM_INSTALMENT_NUMBER	-0.002095	-0.000548	-0.323414	1.000000	0.090286	
DAYS_INSTALMENT	0.003748	0.001191	0.130244	0.090286	1.000000	
DAYS_ENTRY_PAYMENT	0.003734	0.001215	0.128124	0.094305	0.999491	
AMT_INSTALMENT	0.002042	-0.000226	0.168109	-0.089640	0.125985	
AMT_PAYMENT	0.001887	-0.000124	0.177176	-0.087664	0.127018	

Missing data for installments_payments

In [36]:
```
percent = (datasets["installments_payments"].isnull().sum()/datasets["installments_payments"].isnull().count()*
sum_missing = datasets["installments_payments"].isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Test Missing Coun
missing_application_train_data.head(20)
```

Out[36]:

	Percent	Test Missing Count
DAYS_ENTRY_PAYMENT	0.02	2905
AMT_PAYMENT	0.02	2905
SK_ID_PREV	0.00	0
SK_ID_CURR	0.00	0
NUM_INSTALMENT_VERSION	0.00	0
NUM_INSTALMENT_NUMBER	0.00	0
DAYS_INSTALMENT	0.00	0
AMT_INSTALMENT	0.00	0

Phase 4: Multi Layer Perceptron Models (Ran on Google Colab and on a mac)

Imports

In [1]:
```
# Import necessary libraries for data preprocessing
import os
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from pandas.plotting import scatter_matrix

# Import necessary libraries for data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Import necessary libraries for logistic regression
from sklearn.linear_model import LogisticRegression

# Import necessary libraries for model selection and evaluation
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import auc, accuracy_score, confusion_matrix, f1_score, log_loss, classification_report, r

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Import necessary libraries for building and training neural network
import time
from datetime import datetime
import json
import pickle
import copy
```

```
import torch
import tensorflow as tf
import torch.nn as nn
import torch.nn.functional as func
from torch.nn.functional import binary_cross_entropy
import torch.optim as optim
from torch.optim import Adam
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras import layers
from tensorflow.keras.callbacks import LearningRateScheduler
```

In [2]:
```
# Import necessary libraries
import time
from datetime import datetime
import json
import pickle
import copy
import warnings

import numpy as np
import pandas as pd
import torch
import tensorflow as tf
import torch.nn as nn
import torch.nn.functional as func
from torch.nn.functional import binary_cross_entropy
import torch.optim as optim
from torch.optim import Adam
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline, FeatureUnion, make_pipeline
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import auc, accuracy_score, confusion_matrix, f1_score, log_loss, classification_report, r

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras import layers
from tensorflow.keras.callbacks import LearningRateScheduler

# Ignore warnings
warnings.filterwarnings('ignore')

# our import script contains code for data preprocessing and a neural network model.
```

In [3]:
```
DATA_DIR = "home-credit-default-risk"   #same level as course repo in the data directory
#DATA_DIR = os.path.join('./ddddd/')
#!mkdir DATA_DIR
```

In [4]:
```
def load_data(in_path, name):
    df = pd.read_csv(in_path)
    print(f"{name}: shape is {df.shape}")
    print(df.info())
    display(df.head(5))
    return df

datasets={}  # lets store the datasets in a dictionary so we can keep track of them easily
ds_name = 'application_train'
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)

datasets['application_train'].shape
```

```
application_train: shape is (307511, 122)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
None
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_
0	100002	1	Cash loans	M	N	Y	0	2
1	100003	0	Cash loans	F	N	N	0	2
2	100004	0	Revolving loans	M	Y	Y	0	
3	100006	0	Cash loans	F	N	Y	0	1
4	100007	0	Cash loans	M	N	Y	0	1

5 rows × 122 columns

Out[4]: (307511, 122)

In [5]:
```
ds_name = 'application_test'
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)


ds_name = 'bureau'
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)

ds_name = 'previous_application'
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)

ds_name = 'installments_payments'
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)
```

```
application_test: shape is (48744, 121)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48744 entries, 0 to 48743
Columns: 121 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(40), object(16)
memory usage: 45.0+ MB
None
```

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	A
0	100001	Cash loans	F	N	Y	0	135000.0	
1	100005	Cash loans	M	N	Y	0	99000.0	
2	100013	Cash loans	M	Y	Y	0	202500.0	
3	100028	Cash loans	F	N	Y	2	315000.0	
4	100038	Cash loans	M	Y	N	1	180000.0	

5 rows × 121 columns

```
bureau: shape is (1716428, 17)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1716428 entries, 0 to 1716427
Data columns (total 17 columns):
 #   Column                  Dtype
---  ------                  -----
 0   SK_ID_CURR              int64
 1   SK_ID_BUREAU            int64
 2   CREDIT_ACTIVE           object
 3   CREDIT_CURRENCY         object
 4   DAYS_CREDIT             int64
 5   CREDIT_DAY_OVERDUE      int64
 6   DAYS_CREDIT_ENDDATE     float64
 7   DAYS_ENDDATE_FACT       float64
 8   AMT_CREDIT_MAX_OVERDUE  float64
 9   CNT_CREDIT_PROLONG      int64
 10  AMT_CREDIT_SUM          float64
 11  AMT_CREDIT_SUM_DEBT     float64
 12  AMT_CREDIT_SUM_LIMIT    float64
 13  AMT_CREDIT_SUM_OVERDUE  float64
 14  CREDIT_TYPE             object
 15  DAYS_CREDIT_UPDATE      int64
 16  AMT_ANNUITY             float64
dtypes: float64(8), int64(6), object(3)
memory usage: 222.6+ MB
None
```

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	D
0	215354	5714462	Closed	currency 1	-497	0	-153.0	
1	215354	5714463	Active	currency 1	-208	0	1075.0	
2	215354	5714464	Active	currency 1	-203	0	528.0	
3	215354	5714465	Active	currency 1	-203	0	NaN	
4	215354	5714466	Active	currency 1	-629	0	1197.0	

```
previous_application: shape is (1670214, 37)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 37 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   SK_ID_PREV                   1670214 non-null  int64
 1   SK_ID_CURR                   1670214 non-null  int64
 2   NAME_CONTRACT_TYPE           1670214 non-null  object
 3   AMT_ANNUITY                  1297979 non-null  float64
 4   AMT_APPLICATION              1670214 non-null  float64
 5   AMT_CREDIT                   1670213 non-null  float64
 6   AMT_DOWN_PAYMENT             774370 non-null   float64
 7   AMT_GOODS_PRICE              1284699 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object
 9   HOUR_APPR_PROCESS_START      1670214 non-null  int64
 10  FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object
 11  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64
 12  RATE_DOWN_PAYMENT            774370 non-null   float64
 13  RATE_INTEREST_PRIMARY        5951 non-null     float64
 14  RATE_INTEREST_PRIVILEGED     5951 non-null     float64
 15  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object
 16  NAME_CONTRACT_STATUS         1670214 non-null  object
 17  DAYS_DECISION                1670214 non-null  int64
 18  NAME_PAYMENT_TYPE            1670214 non-null  object
 19  CODE_REJECT_REASON           1670214 non-null  object
 20  NAME_TYPE_SUITE              849809 non-null   object
 21  NAME_CLIENT_TYPE             1670214 non-null  object
 22  NAME_GOODS_CATEGORY          1670214 non-null  object
 23  NAME_PORTFOLIO               1670214 non-null  object
 24  NAME_PRODUCT_TYPE            1670214 non-null  object
 25  CHANNEL_TYPE                 1670214 non-null  object
 26  SELLERPLACE_AREA             1670214 non-null  int64
 27  NAME_SELLER_INDUSTRY         1670214 non-null  object
 28  CNT_PAYMENT                  1297984 non-null  float64
 29  NAME_YIELD_GROUP             1670214 non-null  object
 30  PRODUCT_COMBINATION          1669868 non-null  object
 31  DAYS_FIRST_DRAWING           997149 non-null   float64
 32  DAYS_FIRST_DUE               997149 non-null   float64
 33  DAYS_LAST_DUE_1ST_VERSION    997149 non-null   float64
 34  DAYS_LAST_DUE                997149 non-null   float64
 35  DAYS_TERMINATION             997149 non-null   float64
 36  NFLAG_INSURED_ON_APPROVAL    997149 non-null   float64
dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
None
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOO
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	0.0	
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	NaN	
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	NaN	
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	NaN	
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	NaN	

5 rows × 37 columns

```
installments_payments: shape is (13605401, 8)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Dtype
---  ------                  -----
 0   SK_ID_PREV              int64
 1   SK_ID_CURR              int64
 2   NUM_INSTALMENT_VERSION  float64
 3   NUM_INSTALMENT_NUMBER   int64
 4   DAYS_INSTALMENT         float64
 5   DAYS_ENTRY_PAYMENT      float64
 6   AMT_INSTALMENT          float64
 7   AMT_PAYMENT             float64
dtypes: float64(5), int64(3)
memory usage: 830.4 MB
None
```

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT	AMT
0	1054186	161674	1.0	6	-1180.0	-1187.0	
1	1330831	151639	0.0	34	-2156.0	-2156.0	
2	2085231	193053	2.0	1	-63.0	-63.0	
3	2452527	199697	1.0	3	-2418.0	-2426.0	
4	2714724	167756	1.0	2	-1383.0	-1366.0	

In [6]:
```
%%time
```

```
ds_names = ("application_train", "application_test", "bureau","bureau_balance","credit_card_balance","installme
            "previous_application","POS_CASH_balance")

for ds_name in ds_names:
    datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)
```

```
application_train: shape is (307511, 122)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
None
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_
0	100002	1	Cash loans	M	N	Y	0	2
1	100003	0	Cash loans	F	N	N	0	2
2	100004	0	Revolving loans	M	Y	Y	0	
3	100006	0	Cash loans	F	N	Y	0	1
4	100007	0	Cash loans	M	N	Y	0	1

5 rows × 122 columns

```
application_test: shape is (48744, 121)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48744 entries, 0 to 48743
Columns: 121 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(40), object(16)
memory usage: 45.0+ MB
None
```

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	A
0	100001	Cash loans	F	N	Y	0	135000.0	
1	100005	Cash loans	M	N	Y	0	99000.0	
2	100013	Cash loans	M	Y	Y	0	202500.0	
3	100028	Cash loans	F	N	Y	2	315000.0	
4	100038	Cash loans	M	Y	N	1	180000.0	

5 rows × 121 columns

```
bureau: shape is (1716428, 17)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1716428 entries, 0 to 1716427
Data columns (total 17 columns):
 #   Column                  Dtype
---  ------                  -----
 0   SK_ID_CURR              int64
 1   SK_ID_BUREAU            int64
 2   CREDIT_ACTIVE           object
 3   CREDIT_CURRENCY         object
 4   DAYS_CREDIT             int64
 5   CREDIT_DAY_OVERDUE      int64
 6   DAYS_CREDIT_ENDDATE     float64
 7   DAYS_ENDDATE_FACT       float64
 8   AMT_CREDIT_MAX_OVERDUE  float64
 9   CNT_CREDIT_PROLONG      int64
 10  AMT_CREDIT_SUM          float64
 11  AMT_CREDIT_SUM_DEBT     float64
 12  AMT_CREDIT_SUM_LIMIT    float64
 13  AMT_CREDIT_SUM_OVERDUE  float64
 14  CREDIT_TYPE             object
 15  DAYS_CREDIT_UPDATE      int64
 16  AMT_ANNUITY             float64
dtypes: float64(8), int64(6), object(3)
memory usage: 222.6+ MB
None
```

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	D
0	215354	5714462	Closed	currency 1	-497	0	-153.0	
1	215354	5714463	Active	currency 1	-208	0	1075.0	
2	215354	5714464	Active	currency 1	-203	0	528.0	
3	215354	5714465	Active	currency 1	-203	0	NaN	
4	215354	5714466	Active	currency 1	-629	0	1197.0	

```
bureau_balance: shape is (27299925, 3)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27299925 entries, 0 to 27299924
Data columns (total 3 columns):
 #   Column          Dtype
---  ------          -----
 0   SK_ID_BUREAU    int64
 1   MONTHS_BALANCE  int64
 2   STATUS          object
dtypes: int64(2), object(1)
memory usage: 624.8+ MB
None
```

	SK_ID_BUREAU	MONTHS_BALANCE	STATUS
0	5715448	0	C
1	5715448	-1	C
2	5715448	-2	C
3	5715448	-3	C
4	5715448	-4	C

```
credit_card_balance: shape is (3840312, 23)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3840312 entries, 0 to 3840311
Data columns (total 23 columns):
 #   Column                       Dtype
---  ------                       -----
 0   SK_ID_PREV                   int64
 1   SK_ID_CURR                   int64
 2   MONTHS_BALANCE               int64
 3   AMT_BALANCE                  float64
 4   AMT_CREDIT_LIMIT_ACTUAL      int64
 5   AMT_DRAWINGS_ATM_CURRENT     float64
 6   AMT_DRAWINGS_CURRENT         float64
 7   AMT_DRAWINGS_OTHER_CURRENT   float64
 8   AMT_DRAWINGS_POS_CURRENT     float64
 9   AMT_INST_MIN_REGULARITY      float64
 10  AMT_PAYMENT_CURRENT          float64
 11  AMT_PAYMENT_TOTAL_CURRENT    float64
 12  AMT_RECEIVABLE_PRINCIPAL     float64
 13  AMT_RECIVABLE                float64
 14  AMT_TOTAL_RECEIVABLE         float64
 15  CNT_DRAWINGS_ATM_CURRENT     float64
 16  CNT_DRAWINGS_CURRENT         int64
 17  CNT_DRAWINGS_OTHER_CURRENT   float64
 18  CNT_DRAWINGS_POS_CURRENT     float64
 19  CNT_INSTALMENT_MATURE_CUM    float64
 20  NAME_CONTRACT_STATUS         object
 21  SK_DPD                       int64
 22  SK_DPD_DEF                   int64
dtypes: float64(15), int64(7), object(1)
memory usage: 673.9+ MB
None
```

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_DRAWINGS_ATM_CURRENT	AMT_DR
0	2562384	378907	-6	56.970	135000	0.0	
1	2582071	363914	-1	63975.555	45000	2250.0	
2	1740877	371185	-7	31815.225	450000	0.0	
3	1389973	337855	-4	236572.110	225000	2250.0	
4	1891521	126868	-1	453919.455	450000	0.0	

5 rows × 23 columns

```
installments_payments: shape is (13605401, 8)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Dtype
---  ------                  -----
 0   SK_ID_PREV              int64
 1   SK_ID_CURR              int64
 2   NUM_INSTALMENT_VERSION  float64
 3   NUM_INSTALMENT_NUMBER   int64
 4   DAYS_INSTALMENT         float64
 5   DAYS_ENTRY_PAYMENT      float64
 6   AMT_INSTALMENT          float64
 7   AMT_PAYMENT             float64
dtypes: float64(5), int64(3)
memory usage: 830.4 MB
None
```

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT	AMT
0	1054186	161674	1.0	6	-1180.0	-1187.0	
1	1330831	151639	0.0	34	-2156.0	-2156.0	
2	2085231	193053	2.0	1	-63.0	-63.0	
3	2452527	199697	1.0	3	-2418.0	-2426.0	
4	2714724	167756	1.0	2	-1383.0	-1366.0	

```
previous_application: shape is (1670214, 37)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 37 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   SK_ID_PREV                   1670214 non-null  int64
 1   SK_ID_CURR                   1670214 non-null  int64
 2   NAME_CONTRACT_TYPE           1670214 non-null  object
 3   AMT_ANNUITY                  1297979 non-null  float64
 4   AMT_APPLICATION              1670214 non-null  float64
 5   AMT_CREDIT                   1670213 non-null  float64
 6   AMT_DOWN_PAYMENT             774370 non-null   float64
 7   AMT_GOODS_PRICE              1284699 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object
 9   HOUR_APPR_PROCESS_START      1670214 non-null  int64
 10  FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object
 11  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64
 12  RATE_DOWN_PAYMENT            774370 non-null   float64
 13  RATE_INTEREST_PRIMARY        5951 non-null     float64
 14  RATE_INTEREST_PRIVILEGED     5951 non-null     float64
 15  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object
 16  NAME_CONTRACT_STATUS         1670214 non-null  object
 17  DAYS_DECISION                1670214 non-null  int64
 18  NAME_PAYMENT_TYPE            1670214 non-null  object
 19  CODE_REJECT_REASON           1670214 non-null  object
 20  NAME_TYPE_SUITE              849809 non-null   object
 21  NAME_CLIENT_TYPE             1670214 non-null  object
 22  NAME_GOODS_CATEGORY          1670214 non-null  object
 23  NAME_PORTFOLIO               1670214 non-null  object
 24  NAME_PRODUCT_TYPE            1670214 non-null  object
 25  CHANNEL_TYPE                 1670214 non-null  object
 26  SELLERPLACE_AREA             1670214 non-null  int64
 27  NAME_SELLER_INDUSTRY         1670214 non-null  object
 28  CNT_PAYMENT                  1297984 non-null  float64
 29  NAME_YIELD_GROUP             1670214 non-null  object
 30  PRODUCT_COMBINATION          1669868 non-null  object
 31  DAYS_FIRST_DRAWING           997149 non-null   float64
 32  DAYS_FIRST_DUE               997149 non-null   float64
 33  DAYS_LAST_DUE_1ST_VERSION    997149 non-null   float64
 34  DAYS_LAST_DUE                997149 non-null   float64
 35  DAYS_TERMINATION             997149 non-null   float64
 36  NFLAG_INSURED_ON_APPROVAL    997149 non-null   float64
dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
None
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOO
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	0.0	
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	NaN	
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	NaN	
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	NaN	
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	NaN	

5 rows × 37 columns

```
POS_CASH_balance: shape is (10001358, 8)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10001358 entries, 0 to 10001357
Data columns (total 8 columns):
 #   Column                 Dtype
---  ------                 -----
 0   SK_ID_PREV             int64
 1   SK_ID_CURR             int64
 2   MONTHS_BALANCE         int64
 3   CNT_INSTALMENT         float64
 4   CNT_INSTALMENT_FUTURE  float64
 5   NAME_CONTRACT_STATUS   object
 6   SK_DPD                 int64
 7   SK_DPD_DEF             int64
dtypes: float64(2), int64(5), object(1)
memory usage: 610.4+ MB
None
```

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FUTURE	NAME_CONTRACT_STATUS	SK_DPD	SK
0	1803195	182943	-31	48.0	45.0	Active	0	
1	1715348	367990	-33	36.0	35.0	Active	0	
2	1784872	397406	-32	12.0	9.0	Active	0	
3	1903291	269225	-35	48.0	42.0	Active	0	
4	2341044	334279	-35	36.0	35.0	Active	0	

```
CPU times: user 15.4 s, sys: 2.66 s, total: 18 s
Wall time: 18.3 s
```

In [7]:
```python
import pandas as pd

def dataset_summary(dataset, summary_type):
    if summary_type == 'info':
        print("")
        print("The Information of ",dataset + " is given below:")
        return(pd.read_csv(dataset).info())
    elif summary_type == 'head':
        print("")
        print("The head of :", dataset + " is given below:")
        return(display(pd.read_csv(dataset).head()))
    elif summary_type == 'tail':
        print("")
        print("The tail of :", dataset + " is given below:")
        return(display(pd.read_csv(dataset).tail()))
    elif summary_type == 'shape':
        print("")
        print("The shape of :", dataset + " is given below:")
        return(display(pd.read_csv(dataset).shape))
    elif summary_type == 'numerical_feat':
        print("")
        print("Below are the numerical features of :", dataset)
        return(display(pd.read_csv(dataset).describe(include = None)))
    elif summary_type == 'categorical_feat':
        print("")
        print("Below are the categorical features of :", dataset)
        return(display(pd.read_csv(dataset).describe(include = 'object')))
    elif summary_type == 'features':
        print("")
        print("Below are the total described features of :", dataset)
        return(display(pd.read_csv(dataset).describe(include = 'all')))
    elif summary_type == 'describe':
        print("")
        print("The decription of :", dataset + " is given below:")
        return(display(pd.read_csv(dataset).describe()))
    elif summary_type == 'datatype_count':
        print("")
        print("The datatype counts of :", dataset + " is given below:")
        return(pd.read_csv(dataset).dtypes.value_counts())
    elif summary_type == 'value_counts':
        print("")
        print("The value count of :", dataset + " is given below:")
        return(display(pd.read_csv(dataset).value_counts))
    else:
        print("Invalid summary_type")
```

In [8]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
def Missing_Plot(dataset):
    plt.figure(figsize=(210,50))
    sns.displot(
    data=datasets[dataset].iloc[: ,20 :60].isna().melt(value_name="missing"),
    y="variable",
    hue="missing",
    multiple="fill",
    aspect=3
    ).set(title='Missing Values Plot')
Missing_Plot("application_test")
```

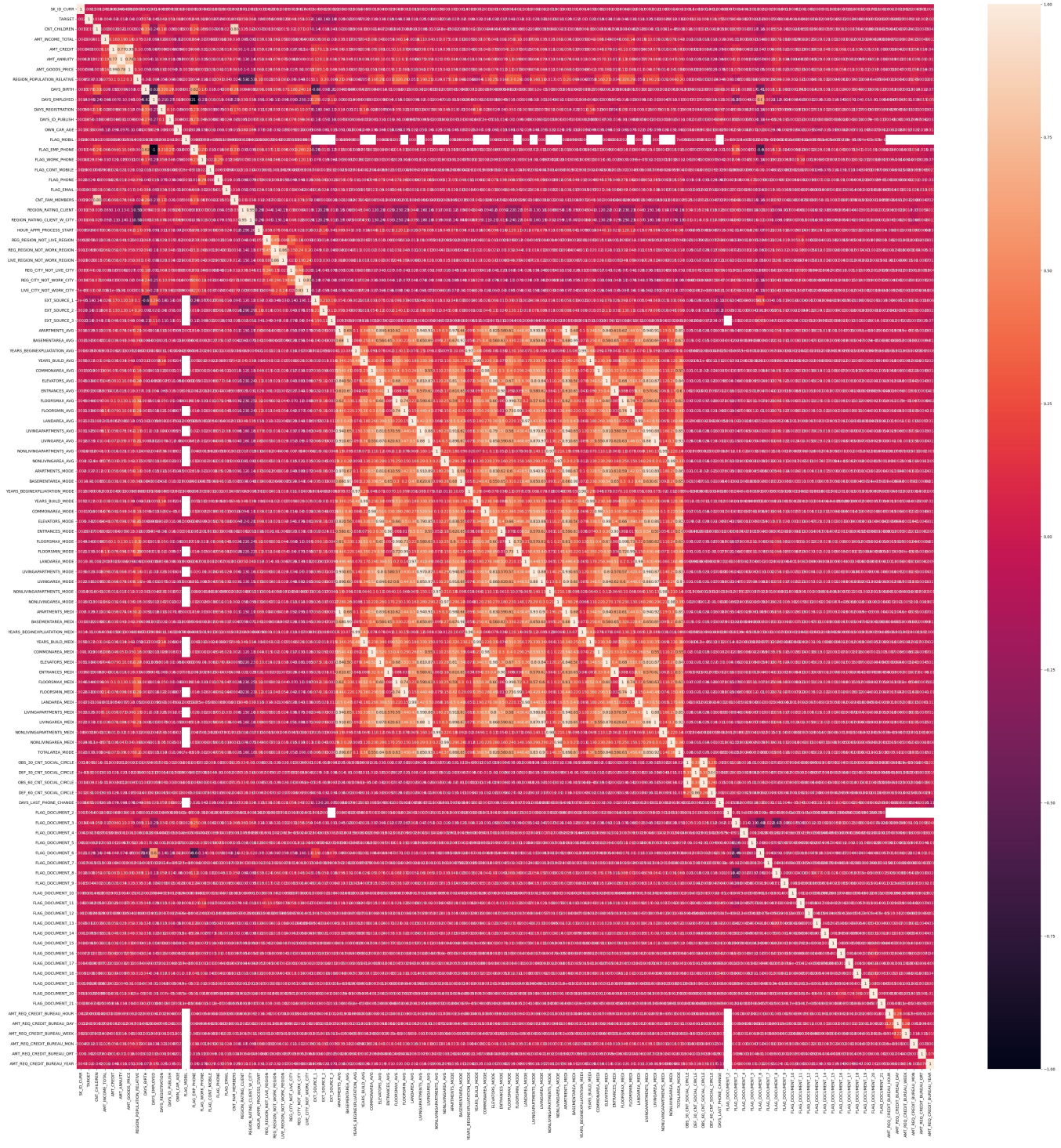
```
<Figure size 21000x5000 with 0 Axes>
```

In [9]:
```
correlations = datasets["application_train"].corr()['TARGET'].sort_values(ascending= True)
print('Most Positive Correlations:\n',correlations.tail(40))
print('\n\n\nMost Negative Correlations:\n',correlations.head(40))
```

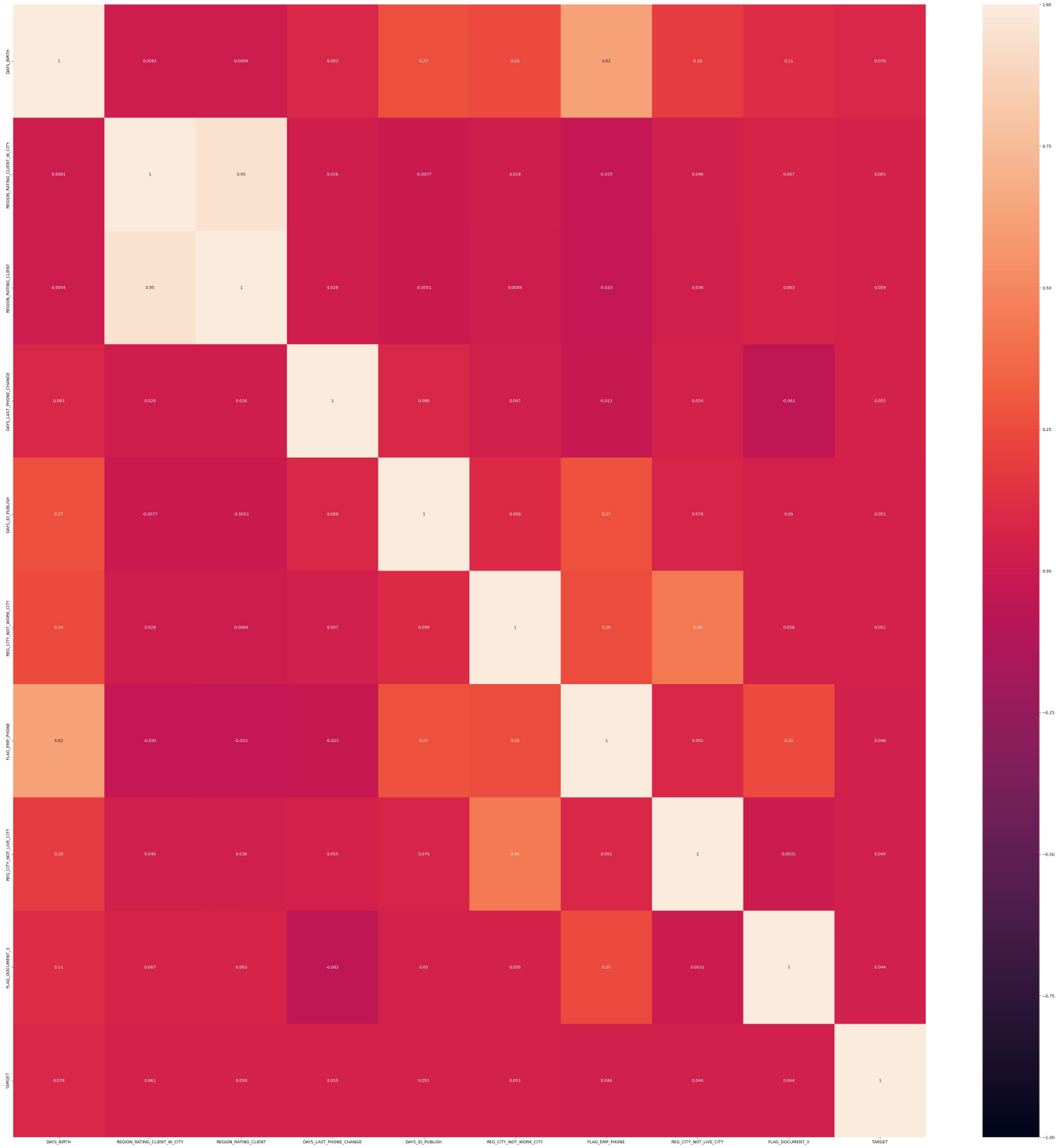
```
Most Positive Correlations:
 AMT_REQ_CREDIT_BUREAU_QRT     -0.002022
FLAG_EMAIL                     -0.001758
NONLIVINGAPARTMENTS_MODE       -0.001557
FLAG_DOCUMENT_7                -0.001520
FLAG_DOCUMENT_10               -0.001414
FLAG_DOCUMENT_19               -0.001358
FLAG_DOCUMENT_12               -0.000756
FLAG_DOCUMENT_5                -0.000316
FLAG_DOCUMENT_20                0.000215
FLAG_CONT_MOBILE                0.000370
FLAG_MOBIL                      0.000534
AMT_REQ_CREDIT_BUREAU_WEEK      0.000788
AMT_REQ_CREDIT_BUREAU_HOUR      0.000930
AMT_REQ_CREDIT_BUREAU_DAY       0.002704
LIVE_REGION_NOT_WORK_REGION     0.002819
FLAG_DOCUMENT_21                0.003709
FLAG_DOCUMENT_2                 0.005417
REG_REGION_NOT_LIVE_REGION      0.005576
REG_REGION_NOT_WORK_REGION      0.006942
OBS_60_CNT_SOCIAL_CIRCLE        0.009022
OBS_30_CNT_SOCIAL_CIRCLE        0.009131
CNT_FAM_MEMBERS                 0.009308
CNT_CHILDREN                    0.019187
AMT_REQ_CREDIT_BUREAU_YEAR      0.019930
FLAG_WORK_PHONE                 0.028524
DEF_60_CNT_SOCIAL_CIRCLE        0.031276
DEF_30_CNT_SOCIAL_CIRCLE        0.032248
LIVE_CITY_NOT_WORK_CITY         0.032518
OWN_CAR_AGE                     0.037612
DAYS_REGISTRATION               0.041975
FLAG_DOCUMENT_3                 0.044346
REG_CITY_NOT_LIVE_CITY          0.044395
FLAG_EMP_PHONE                  0.045982
REG_CITY_NOT_WORK_CITY          0.050994
DAYS_ID_PUBLISH                 0.051457
DAYS_LAST_PHONE_CHANGE          0.055218
REGION_RATING_CLIENT            0.058899
REGION_RATING_CLIENT_W_CITY     0.060893
DAYS_BIRTH                      0.078239
TARGET                          1.000000
Name: TARGET, dtype: float64


Most Negative Correlations:
 EXT_SOURCE_3                  -0.178919
EXT_SOURCE_2                   -0.160472
EXT_SOURCE_1                   -0.155317
DAYS_EMPLOYED                  -0.044932
FLOORSMAX_AVG                  -0.044003
FLOORSMAX_MEDI                 -0.043768
FLOORSMAX_MODE                 -0.043226
AMT_GOODS_PRICE                -0.039645
REGION_POPULATION_RELATIVE     -0.037227
ELEVATORS_AVG                  -0.034199
ELEVATORS_MEDI                 -0.033863
FLOORSMIN_AVG                  -0.033614
FLOORSMIN_MEDI                 -0.033394
LIVINGAREA_AVG                 -0.032997
LIVINGAREA_MEDI                -0.032739
FLOORSMIN_MODE                 -0.032698
TOTALAREA_MODE                 -0.032596
ELEVATORS_MODE                 -0.032131
LIVINGAREA_MODE                -0.030685
AMT_CREDIT                     -0.030369
APARTMENTS_AVG                 -0.029498
APARTMENTS_MEDI                -0.029184
FLAG_DOCUMENT_6                -0.028602
APARTMENTS_MODE                -0.027284
LIVINGAPARTMENTS_AVG           -0.025031
LIVINGAPARTMENTS_MEDI          -0.024621
HOUR_APPR_PROCESS_START        -0.024166
FLAG_PHONE                     -0.023806
LIVINGAPARTMENTS_MODE          -0.023393
BASEMENTAREA_AVG               -0.022746
YEARS_BUILD_MEDI               -0.022326
YEARS_BUILD_AVG                -0.022149
BASEMENTAREA_MEDI              -0.022081
YEARS_BUILD_MODE               -0.022068
BASEMENTAREA_MODE              -0.019952
ENTRANCES_AVG                  -0.019172
ENTRANCES_MEDI                 -0.019025
COMMONAREA_MEDI                -0.018573
COMMONAREA_AVG                 -0.018550
ENTRANCES_MODE                 -0.017387
Name: TARGET, dtype: float64
```

In [10]:
```
plt.figure(figsize = (50,50))
```

```
corrMap = sns.heatmap(datasets["application_train"].corr(), vmin=-1, vmax = 1, annot=True)
```

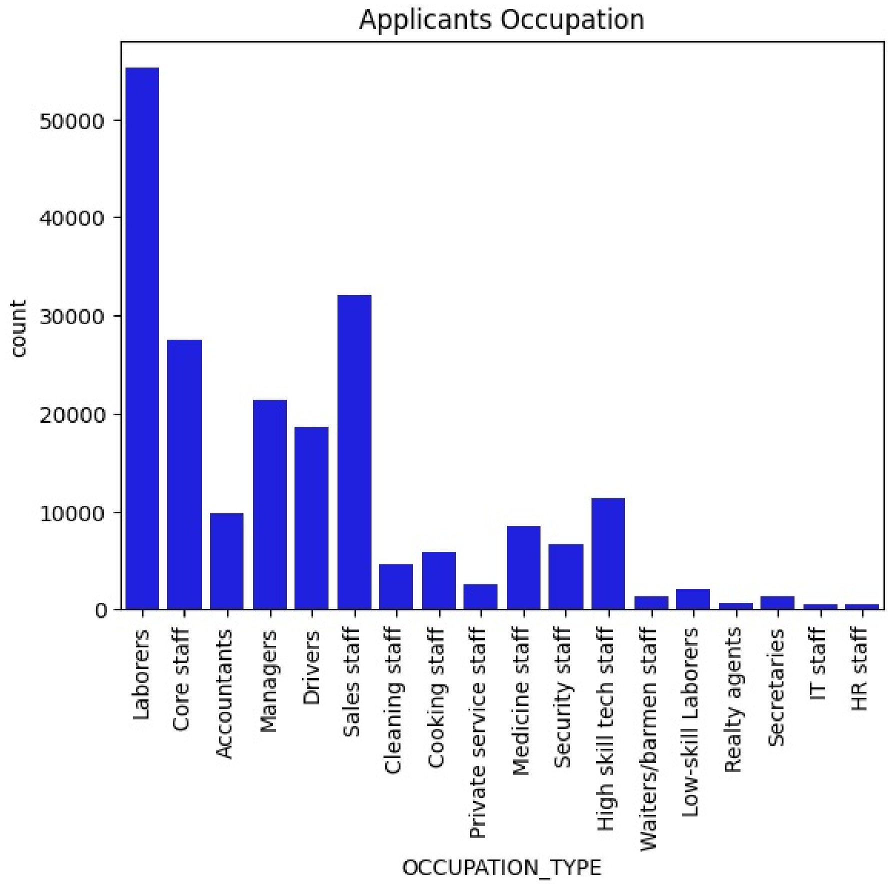


In [11]:
```
# Correlation map of highly positive correlated features of application train to TARGET
plt.figure(figsize = (50,50))
corr_cols = ['DAYS_BIRTH', 'REGION_RATING_CLIENT_W_CITY','REGION_RATING_CLIENT','DAYS_LAST_PHONE_CHANGE', 'DAYS
             'REG_CITY_NOT_WORK_CITY','FLAG_EMP_PHONE','REG_CITY_NOT_LIVE_CITY', 'FLAG_DOCUMENT_3', 'TARGET']
corrMap = sns.heatmap(datasets["application_train"][corr_cols].corr(), vmin=-1, vmax=1, annot=True)
```



In [12]:

```
#Applicants Age
plt.hist(datasets["application_train"]['DAYS_BIRTH'] / -365, edgecolor = 'k', bins = 30)
plt.title('Age of Client'); plt.xlabel('Age (years)'); plt.ylabel('Count');

sns.countplot(x='OCCUPATION_TYPE', data=datasets["application_train"], color='Blue');
plt.title('Applicants Occupation');
plt.xticks(rotation=90);
```

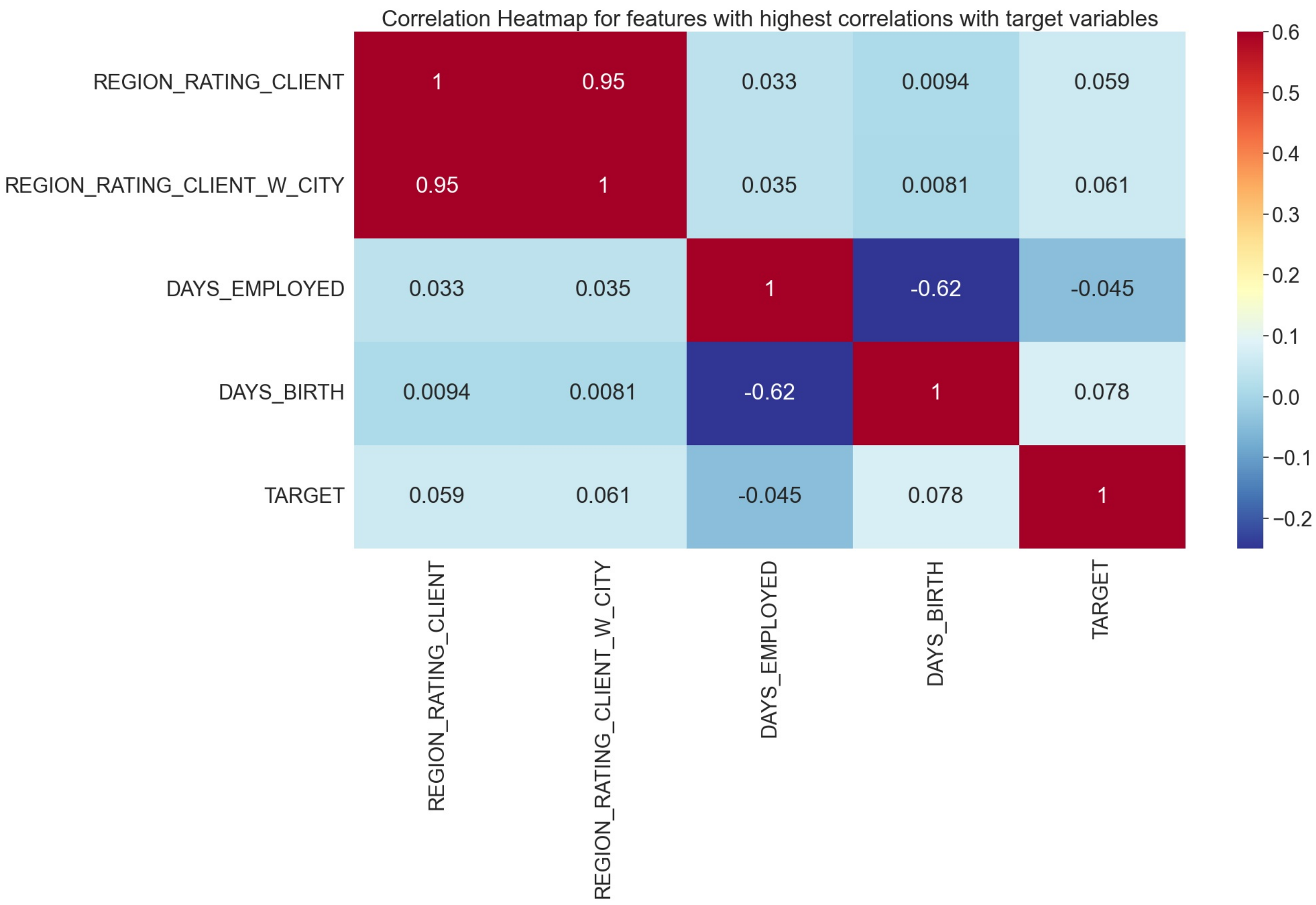


In [13]:

```
most_corr=datasets["application_train"][['REGION_RATING_CLIENT',
                                           'REGION_RATING_CLIENT_W_CITY','DAYS_EMPLOYED','DAYS_BIRTH','TARGET']]
most_corr_corr = most_corr.corr()

sns.set_style("dark")
sns.set_context("notebook", font_scale=2.0, rc={"lines.linewidth": 1.0})
fig, axes = plt.subplots(figsize = (20,10),sharey=True)
sns.heatmap(most_corr_corr,cmap=plt.cm.RdYlBu_r,vmin=-0.25,vmax=0.6,annot=True)
plt.title('Correlation Heatmap for features with highest correlations with target variables')
```

Out[13]:

```
Text(0.5, 1.0, 'Correlation Heatmap for features with highest correlations with target variables')
```



FEATURE ENGINEERING (Carryover from phase 3)

In [14]:
```
import os

def load_data(in_path, name):
        df = pd.read_csv(in_path)
        print(f"{name}: shape is {df.shape}")
        print(df.info())
        display(df.head(5))
        return df
datasets={}
ds_name = 'application_train'
DATA_DIR=f"/Users/deepak/Desktop/AML/home-credit-default-risk/"
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)
datasets['application_train'].shape
```

```
application_train: shape is (307511, 122)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
None
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_
0	100002	1	Cash loans	M	N	Y	0	2
1	100003	0	Cash loans	F	N	N	0	2
2	100004	0	Revolving loans	M	Y	Y	0	
3	100006	0	Cash loans	F	N	Y	0	1
4	100007	0	Cash loans	M	N	Y	0	1

5 rows × 122 columns

Out[14]:
```
(307511, 122)
```

In [15]:
```
ds_name = 'application_test'
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)
```

```
application_test: shape is (48744, 121)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48744 entries, 0 to 48743
Columns: 121 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(40), object(16)
memory usage: 45.0+ MB
None
```

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	A
0	100001	Cash loans	F	N	Y	0	135000.0	
1	100005	Cash loans	M	N	Y	0	99000.0	
2	100013	Cash loans	M	Y	Y	0	202500.0	
3	100028	Cash loans	F	N	Y	2	315000.0	
4	100038	Cash loans	M	Y	N	1	180000.0	

5 rows × 121 columns

In [16]:
```
%%time
ds_names = ("application_train", "application_test", "bureau","bureau_balance","credit_card_balance","installme
            "previous_application","POS_CASH_balance")

for ds_name in ds_names:
    datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)
```

```
application_train: shape is (307511, 122)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
None
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_
0	100002	1	Cash loans	M	N	Y	0	2
1	100003	0	Cash loans	F	N	N	0	2
2	100004	0	Revolving loans	M	Y	Y	0	
3	100006	0	Cash loans	F	N	Y	0	1
4	100007	0	Cash loans	M	N	Y	0	1

5 rows × 122 columns

```
application_test: shape is (48744, 121)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48744 entries, 0 to 48743
Columns: 121 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(40), object(16)
memory usage: 45.0+ MB
None
```

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	A
0	100001	Cash loans	F	N	Y	0	135000.0	
1	100005	Cash loans	M	N	Y	0	99000.0	
2	100013	Cash loans	M	Y	Y	0	202500.0	
3	100028	Cash loans	F	N	Y	2	315000.0	
4	100038	Cash loans	M	Y	N	1	180000.0	

5 rows × 121 columns

```
bureau: shape is (1716428, 17)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1716428 entries, 0 to 1716427
Data columns (total 17 columns):
 #   Column                  Dtype
---  ------                  -----
 0   SK_ID_CURR              int64
 1   SK_ID_BUREAU            int64
 2   CREDIT_ACTIVE           object
 3   CREDIT_CURRENCY         object
 4   DAYS_CREDIT             int64
 5   CREDIT_DAY_OVERDUE      int64
 6   DAYS_CREDIT_ENDDATE     float64
 7   DAYS_ENDDATE_FACT       float64
 8   AMT_CREDIT_MAX_OVERDUE  float64
 9   CNT_CREDIT_PROLONG      int64
 10  AMT_CREDIT_SUM          float64
 11  AMT_CREDIT_SUM_DEBT     float64
 12  AMT_CREDIT_SUM_LIMIT    float64
 13  AMT_CREDIT_SUM_OVERDUE  float64
 14  CREDIT_TYPE             object
 15  DAYS_CREDIT_UPDATE      int64
 16  AMT_ANNUITY             float64
dtypes: float64(8), int64(6), object(3)
memory usage: 222.6+ MB
None
```

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	D
0	215354	5714462	Closed	currency 1	-497	0	-153.0	
1	215354	5714463	Active	currency 1	-208	0	1075.0	
2	215354	5714464	Active	currency 1	-203	0	528.0	
3	215354	5714465	Active	currency 1	-203	0	NaN	
4	215354	5714466	Active	currency 1	-629	0	1197.0	

```
bureau_balance: shape is (27299925, 3)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27299925 entries, 0 to 27299924
Data columns (total 3 columns):
 #   Column          Dtype
---  ------          -----
 0   SK_ID_BUREAU    int64
 1   MONTHS_BALANCE  int64
 2   STATUS          object
dtypes: int64(2), object(1)
memory usage: 624.8+ MB
None
```

	SK_ID_BUREAU	MONTHS_BALANCE	STATUS
0	5715448	0	C
1	5715448	-1	C
2	5715448	-2	C
3	5715448	-3	C
4	5715448	-4	C

```
credit_card_balance: shape is (3840312, 23)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3840312 entries, 0 to 3840311
Data columns (total 23 columns):
 #   Column                      Dtype  
---  ------                      -----  
 0   SK_ID_PREV                  int64  
 1   SK_ID_CURR                  int64  
 2   MONTHS_BALANCE              int64  
 3   AMT_BALANCE                 float64
 4   AMT_CREDIT_LIMIT_ACTUAL     int64  
 5   AMT_DRAWINGS_ATM_CURRENT    float64
 6   AMT_DRAWINGS_CURRENT        float64
 7   AMT_DRAWINGS_OTHER_CURRENT  float64
 8   AMT_DRAWINGS_POS_CURRENT    float64
 9   AMT_INST_MIN_REGULARITY     float64
 10  AMT_PAYMENT_CURRENT         float64
 11  AMT_PAYMENT_TOTAL_CURRENT   float64
 12  AMT_RECEIVABLE_PRINCIPAL    float64
 13  AMT_RECIVABLE               float64
 14  AMT_TOTAL_RECEIVABLE        float64
 15  CNT_DRAWINGS_ATM_CURRENT    float64
 16  CNT_DRAWINGS_CURRENT        int64  
 17  CNT_DRAWINGS_OTHER_CURRENT  float64
 18  CNT_DRAWINGS_POS_CURRENT    float64
 19  CNT_INSTALMENT_MATURE_CUM   float64
 20  NAME_CONTRACT_STATUS        object 
 21  SK_DPD                      int64  
 22  SK_DPD_DEF                  int64  
dtypes: float64(15), int64(7), object(1)
memory usage: 673.9+ MB
None
```

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_DRAWINGS_ATM_CURRENT	AMT_DR
0	2562384	378907	-6	56.970	135000	0.0	
1	2582071	363914	-1	63975.555	45000	2250.0	
2	1740877	371185	-7	31815.225	450000	0.0	
3	1389973	337855	-4	236572.110	225000	2250.0	
4	1891521	126868	-1	453919.455	450000	0.0	

5 rows × 23 columns

```
installments_payments: shape is (13605401, 8)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Dtype  
---  ------                  -----  
 0   SK_ID_PREV              int64  
 1   SK_ID_CURR              int64  
 2   NUM_INSTALMENT_VERSION  float64
 3   NUM_INSTALMENT_NUMBER   int64  
 4   DAYS_INSTALMENT         float64
 5   DAYS_ENTRY_PAYMENT      float64
 6   AMT_INSTALMENT          float64
 7   AMT_PAYMENT             float64
dtypes: float64(5), int64(3)
memory usage: 830.4 MB
None
```

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT	AM
0	1054186	161674	1.0	6	-1180.0	-1187.0	
1	1330831	151639	0.0	34	-2156.0	-2156.0	
2	2085231	193053	2.0	1	-63.0	-63.0	
3	2452527	199697	1.0	3	-2418.0	-2426.0	
4	2714724	167756	1.0	2	-1383.0	-1366.0	

```
previous_application: shape is (1670214, 37)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 37 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   SK_ID_PREV                   1670214 non-null  int64
 1   SK_ID_CURR                   1670214 non-null  int64
 2   NAME_CONTRACT_TYPE           1670214 non-null  object
 3   AMT_ANNUITY                  1297979 non-null  float64
 4   AMT_APPLICATION              1670214 non-null  float64
 5   AMT_CREDIT                   1670213 non-null  float64
 6   AMT_DOWN_PAYMENT             774370 non-null   float64
 7   AMT_GOODS_PRICE              1284699 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object
 9   HOUR_APPR_PROCESS_START      1670214 non-null  int64
 10  FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object
 11  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64
 12  RATE_DOWN_PAYMENT            774370 non-null   float64
 13  RATE_INTEREST_PRIMARY        5951 non-null     float64
 14  RATE_INTEREST_PRIVILEGED     5951 non-null     float64
 15  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object
 16  NAME_CONTRACT_STATUS         1670214 non-null  object
 17  DAYS_DECISION                1670214 non-null  int64
 18  NAME_PAYMENT_TYPE            1670214 non-null  object
 19  CODE_REJECT_REASON           1670214 non-null  object
 20  NAME_TYPE_SUITE              849809 non-null   object
 21  NAME_CLIENT_TYPE             1670214 non-null  object
 22  NAME_GOODS_CATEGORY          1670214 non-null  object
 23  NAME_PORTFOLIO               1670214 non-null  object
 24  NAME_PRODUCT_TYPE            1670214 non-null  object
 25  CHANNEL_TYPE                 1670214 non-null  object
 26  SELLERPLACE_AREA             1670214 non-null  int64
 27  NAME_SELLER_INDUSTRY         1670214 non-null  object
 28  CNT_PAYMENT                  1297984 non-null  float64
 29  NAME_YIELD_GROUP             1670214 non-null  object
 30  PRODUCT_COMBINATION          1669868 non-null  object
 31  DAYS_FIRST_DRAWING           997149 non-null   float64
 32  DAYS_FIRST_DUE               997149 non-null   float64
 33  DAYS_LAST_DUE_1ST_VERSION    997149 non-null   float64
 34  DAYS_LAST_DUE                997149 non-null   float64
 35  DAYS_TERMINATION             997149 non-null   float64
 36  NFLAG_INSURED_ON_APPROVAL    997149 non-null   float64
dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
None
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOO
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	0.0	
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	NaN	
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	NaN	
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	NaN	
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	NaN	

5 rows × 37 columns

```
POS_CASH_balance: shape is (10001358, 8)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10001358 entries, 0 to 10001357
Data columns (total 8 columns):
 #   Column                 Dtype
---  ------                 -----
 0   SK_ID_PREV             int64
 1   SK_ID_CURR             int64
 2   MONTHS_BALANCE         int64
 3   CNT_INSTALMENT         float64
 4   CNT_INSTALMENT_FUTURE  float64
 5   NAME_CONTRACT_STATUS   object
 6   SK_DPD                 int64
 7   SK_DPD_DEF             int64
dtypes: float64(2), int64(5), object(1)
memory usage: 610.4+ MB
None
```

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FUTURE	NAME_CONTRACT_STATUS	SK_DPD	SK
0	1803195	182943	-31	48.0	45.0	Active	0	
1	1715348	367990	-33	36.0	35.0	Active	0	
2	1784872	397406	-32	12.0	9.0	Active	0	
3	1903291	269225	-35	48.0	42.0	Active	0	
4	2341044	334279	-35	36.0	35.0	Active	0	

```
CPU times: user 15.6 s, sys: 2.61 s, total: 18.2 s
Wall time: 18.4 s
```

In [17]:
```
for ds_name in datasets.keys():
    print(f'dataset {ds_name:24}: [ {datasets[ds_name].shape[0]:10,}, {datasets[ds_name].shape[1]}]')
```

```
dataset application_train       : [    307,511, 122]
dataset application_test        : [     48,744, 121]
dataset bureau                  : [  1,716,428, 17]
dataset bureau_balance          : [ 27,299,925, 3]
dataset credit_card_balance     : [  3,840,312, 23]
dataset installments_payments   : [ 13,605,401, 8]
dataset previous_application    : [  1,670,214, 37]
dataset POS_CASH_balance        : [ 10,001,358, 8]
```

Undersampling - Due to class imbalance

In [18]:
```
# Access the 'application_train' dataset from the 'datasets' container
application_train = datasets['application_train']

# Select the minority class instances (TARGET = 1) from the training dataset
minority_application_train = application_train[application_train['TARGET']==1]

# Append a randomly sampled subset of majority class instances (TARGET = 0) to the minority class instances
undersampled_application_train = minority_application_train.append(
    application_train[application_train['TARGET']==0].reset_index(drop=True).sample(n = 75000)
)
```

In [19]:
```
# Assign the undersampled training dataset to a new key in the 'datasets' dictionary
datasets["undersampled_application_train"] = undersampled_application_train

# Count the number of instances in each class
class_distribution = undersampled_application_train['TARGET'].value_counts()

# Print the class distribution
print("Class distribution in the undersampled training dataset:")
print(class_distribution)
```

```
Class distribution in the undersampled training dataset:
0    75000
1    24825
Name: TARGET, dtype: int64
```

In [20]:
```
  # Assuming this is a dictionary where you store your datasets

# Filtering rows with TARGET == 1 and creating a new DataFrame
datasets["undersampled_application_train_2"] = datasets["application_train"][datasets["application_train"].TARG
datasets["undersampled_application_train_2"]['weight'] = 1

# Undersampling Cash loans
num_default_cashloans = len(datasets["undersampled_application_train_2"][(datasets["undersampled_application_tr
df_sample_cash = datasets["application_train"][(datasets["application_train"].NAME_CONTRACT_TYPE == 'Cash loans
df_sample_cash['weight'] = 1

# Undersampling Revolving loans
num_default_revolvingloans = len(datasets["undersampled_application_train_2"][(datasets["undersampled_applicati
df_sample_revolving = datasets["application_train"][(datasets["application_train"].NAME_CONTRACT_TYPE == 'Revol
df_sample_revolving['weight'] = 1

# Combining undersampled cash loans and revolving loans with the initial DataFrame
datasets["undersampled_application_train_2"] = pd.concat([datasets["undersampled_application_train_2"], df_samp

# Check the distribution of the TARGET variable
print(datasets["undersampled_application_train_2"].TARGET.value_counts())
```

```
1    24825
0    24825
Name: TARGET, dtype: int64
```

In [21]:
```
# Assuming this is a dictionary where you store your datasets

# Filtering rows with TARGET == 1 and creating a new DataFrame
undersampled_application_train_2 = datasets["application_train"][datasets["application_train"].TARGET == 1].cop
undersampled_application_train_2['weight'] = 1

# Undersampling Cash loans
num_default_cashloans = len(undersampled_application_train_2[(undersampled_application_train_2.NAME_CONTRACT_TY
df_sample_cash = datasets["application_train"][(datasets["application_train"].NAME_CONTRACT_TYPE == 'Cash loans
df_sample_cash['weight'] = 1

# Undersampling Revolving loans
num_default_revolvingloans = len(undersampled_application_train_2[(undersampled_application_train_2.NAME_CONTRA
df_sample_revolving = datasets["application_train"][(datasets["application_train"].NAME_CONTRACT_TYPE == 'Revol
df_sample_revolving['weight'] = 1

# Combining undersampled cash loans and revolving loans with the initial DataFrame
undersampled_application_train_2 = pd.concat([undersampled_application_train_2, df_sample_cash, df_sample_revol

# Check the distribution of the TARGET variable
```

```
print(undersampled_application_train_2.TARGET.value_counts())
```

```
1    24825
0    24825
Name: TARGET, dtype: int64
```

In [22]:
```
# Create aggregate features (via pipeline)
class FeaturesAggregater(BaseEstimator, TransformerMixin):

    def __init__(self, features=None, agg_needed=["mean"]): # no *args or **kargs self.features = features
        self.agg_needed = agg_needed
        self.agg_op_features = {}
        for f in features:
            self.agg_op_features[f] = self.agg_needed[:]
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        result = X.groupby(["SK_ID_CURR"]).agg(self.agg_op_features)
        df_result = pd.DataFrame()
        for x1, x2 in result.columns:
            new_col = x1 + "_" + x2
            df_result[new_col] = result[x1][x2]
            df_result = df_result.reset_index(level=["SK_ID_CURR"])
            return df_result
```

In [23]:
```
# Access the 'previous_application' dataset from the 'datasets' container and assign it to a variable named 'pr
previous_application_data = datasets["previous_application"]

# Apply the 'isna()' method on the 'previous_application_data' DataFrame to detect missing or null values,
# and then apply the 'sum()' method to count the number of missing values in each column of the DataFrame.
missing_values_count_per_column = previous_application_data.isna().sum()
missing_values_count_per_column
```

Out[23]:
```
SK_ID_PREV                        0
SK_ID_CURR                        0
NAME_CONTRACT_TYPE                0
AMT_ANNUITY                  372235
AMT_APPLICATION                   0
AMT_CREDIT                        1
AMT_DOWN_PAYMENT             895844
AMT_GOODS_PRICE              385515
WEEKDAY_APPR_PROCESS_START        0
HOUR_APPR_PROCESS_START           0
FLAG_LAST_APPL_PER_CONTRACT       0
NFLAG_LAST_APPL_IN_DAY            0
RATE_DOWN_PAYMENT            895844
RATE_INTEREST_PRIMARY       1664263
RATE_INTEREST_PRIVILEGED    1664263
NAME_CASH_LOAN_PURPOSE            0
NAME_CONTRACT_STATUS              0
DAYS_DECISION                     0
NAME_PAYMENT_TYPE                 0
CODE_REJECT_REASON                0
NAME_TYPE_SUITE              820405
NAME_CLIENT_TYPE                  0
NAME_GOODS_CATEGORY               0
NAME_PORTFOLIO                    0
NAME_PRODUCT_TYPE                 0
CHANNEL_TYPE                      0
SELLERPLACE_AREA                  0
NAME_SELLER_INDUSTRY              0
CNT_PAYMENT                  372230
NAME_YIELD_GROUP                  0
PRODUCT_COMBINATION             346
DAYS_FIRST_DRAWING           673065
DAYS_FIRST_DUE               673065
DAYS_LAST_DUE_1ST_VERSION    673065
DAYS_LAST_DUE                673065
DAYS_TERMINATION             673065
NFLAG_INSURED_ON_APPROVAL    673065
dtype: int64
```

In [24]:
```
previous_feature = ["AMT_APPLICATION", "AMT_CREDIT", "AMT_ANNUITY", "approved_credit_ratio", "AMT_ANNUITY_credi
agg_needed = ["min", "max", "mean", "count", "sum"]


agg_needed = ["min", "max", "mean", "count", "sum"]

def previous_feature_aggregation(df, feature, agg_needed):
    df['approved_credit_ratio'] = (df['AMT_APPLICATION']/df['AMT_CREDIT']).replace(np.inf, 0)
    # installment over credit approved ratio
    df['AMT_ANNUITY_credit_ratio'] = (df['AMT_ANNUITY']/df['AMT_CREDIT']).replace(np.inf, 0)
    # total interest payment over credit ratio
    df['Interest_ratio'] = (df['AMT_ANNUITY']/df['AMT_CREDIT']).replace(np.inf, 0)
    #  loan cover ratio
    df['LTV_ratio'] = (df['AMT_CREDIT']/df['AMT_GOODS_PRICE']).replace(np.inf, 0)
    df['approved'] = np.where(df.AMT_CREDIT >0 ,1, 0)

    test_pipeline = make_pipeline(FeaturesAggregater(feature, agg_needed))
```

```
    return(test_pipeline.fit_transform(df))

datasets['previous_application_agg'] = previous_feature_aggregation(datasets["previous_application"], previous_
```

In [25]:
```
datasets["previous_application_agg"].isna().sum()
```

Out[25]:
```
SK_ID_CURR             0
AMT_APPLICATION_min    0
dtype: int64
```

In [26]:
```
datasets["installments_payments"].isna().sum()
```

Out[26]:
```
SK_ID_PREV                  0
SK_ID_CURR                  0
NUM_INSTALMENT_VERSION      0
NUM_INSTALMENT_NUMBER       0
DAYS_INSTALMENT             0
DAYS_ENTRY_PAYMENT       2905
AMT_INSTALMENT              0
AMT_PAYMENT              2905
dtype: int64
```

In [27]:
```
payments_features = ["DAYS_INSTALMENT_DIFF", "AMT_PATMENT_PCT"]

agg_needed = ["mean"]

def payments_feature_aggregation(df, feature, agg_needed):
    df['DAYS_INSTALMENT_DIFF'] = df['DAYS_INSTALMENT'] - df['DAYS_ENTRY_PAYMENT']
    df['AMT_PATMENT_PCT'] = [x/y if (y != 0) & pd.notnull(y) else np.nan for x,y in zip(df.AMT_PAYMENT,df.AMT_I

    test_pipeline = make_pipeline(FeaturesAggregater(feature, agg_needed))
    return(test_pipeline.fit_transform(df))

datasets['installments_payments_agg'] = payments_feature_aggregation(datasets["installments_payments"], payment
```

In [28]:
```
datasets["installments_payments_agg"].isna().sum()
```

Out[28]:
```
SK_ID_CURR                  0
DAYS_INSTALMENT_DIFF_mean   9
dtype: int64
```

In [29]:
```
datasets["credit_card_balance"].isna().sum()
```

Out[29]:
```
SK_ID_PREV                        0
SK_ID_CURR                        0
MONTHS_BALANCE                    0
AMT_BALANCE                       0
AMT_CREDIT_LIMIT_ACTUAL           0
AMT_DRAWINGS_ATM_CURRENT     749816
AMT_DRAWINGS_CURRENT              0
AMT_DRAWINGS_OTHER_CURRENT   749816
AMT_DRAWINGS_POS_CURRENT     749816
AMT_INST_MIN_REGULARITY      305236
AMT_PAYMENT_CURRENT          767988
AMT_PAYMENT_TOTAL_CURRENT         0
AMT_RECEIVABLE_PRINCIPAL          0
AMT_RECIVABLE                     0
AMT_TOTAL_RECEIVABLE              0
CNT_DRAWINGS_ATM_CURRENT     749816
CNT_DRAWINGS_CURRENT              0
CNT_DRAWINGS_OTHER_CURRENT   749816
CNT_DRAWINGS_POS_CURRENT     749816
CNT_INSTALMENT_MATURE_CUM    305236
NAME_CONTRACT_STATUS              0
SK_DPD                            0
SK_DPD_DEF                        0
dtype: int64
```

In [30]:
```
credit_features = [
    "AMT_BALANCE",
    "AMT_DRAWINGS_PCT",
    "AMT_DRAWINGS_ATM_PCT",
    "AMT_DRAWINGS_OTHER_PCT",
    "AMT_DRAWINGS_POS_PCT",
    "AMT_PRINCIPAL_RECEIVABLE_PCT",
    "CNT_DRAWINGS_ATM_CURRENT",
    "CNT_DRAWINGS_CURRENT",
    "CNT_DRAWINGS_OTHER_CURRENT",
    "CNT_DRAWINGS_POS_CURRENT",
    "SK_DPD",
    "SK_DPD_DEF",
]

agg_needed = ["mean"]


def calculate_pct(x, y):
    return x / y if (y != 0) & pd.notnull(y) else np.nan
#def pct(x, y):
```

```
    #return x / y if (y != 0) & pd.notnull(y) else np.nan


def credit_feature_aggregation(df, feature, agg_needed):
    pct_columns = [
        ("AMT_DRAWINGS_CURRENT", "AMT_DRAWINGS_PCT"),
        ("AMT_DRAWINGS_ATM_CURRENT", "AMT_DRAWINGS_ATM_PCT"),
        ("AMT_DRAWINGS_OTHER_CURRENT", "AMT_DRAWINGS_OTHER_PCT"),
        ("AMT_DRAWINGS_POS_CURRENT", "AMT_DRAWINGS_POS_PCT"),
        ("AMT_RECEIVABLE_PRINCIPAL", "AMT_PRINCIPAL_RECEIVABLE_PCT"),
    ]

    for col_x, col_pct in pct_columns:
        df[col_pct] = [calculate_pct(x, y) for x, y in zip(df[col_x], df["AMT_CREDIT_LIMIT_ACTUAL"])]

    pipeline = make_pipeline(FeaturesAggregater(feature, agg_needed))
    return pipeline.fit_transform(df)


datasets["credit_card_balance_agg"] = credit_feature_aggregation(
    datasets["credit_card_balance"], credit_features, agg_needed
)
```

In [31]:
```
datasets["credit_card_balance_agg"].isna().sum()
```

Out[31]:
```
SK_ID_CURR          0
AMT_BALANCE_mean    0
dtype: int64
```

In [32]:
```
datasets.keys()
```

Out[32]:
```
dict_keys(['application_train', 'application_test', 'bureau', 'bureau_balance', 'credit_card_balance', 'install
ments_payments', 'previous_application', 'POS_CASH_balance', 'undersampled_application_train', 'undersampled_ap
plication_train_2', 'previous_application_agg', 'installments_payments_agg', 'credit_card_balance_agg'])
```

In [33]:
```
# Load the train dataset
train_data = datasets["application_train"]

# Compute the distribution of the target variable
target_counts = train_data['TARGET'].value_counts()

# Display the target distribution
print("Target variable distribution:\n")
print(target_counts)
print("\n")

# Compute the percentage of positive and negative examples in the dataset
positive_count = target_counts[1]
negative_count = target_counts[0]
total_count = positive_count + negative_count
positive_percentage = (positive_count / total_count) * 100
negative_percentage = (negative_count / total_count) * 100

# Display the percentages of positive and negative examples
print(f"Percentage of positive examples: {positive_percentage:.2f}%")
print(f"Percentage of negative examples: {negative_percentage:.2f}%")
```

```
Target variable distribution:

0    282686
1     24825
Name: TARGET, dtype: int64


Percentage of positive examples: 8.07%
Percentage of negative examples: 91.93%
```

In [34]:
```
train_dataset= datasets["undersampled_application_train"] #primary dataset

merge_all_data = True

# merge primary table and secondary tables using features based on meta data and  aggregage stats
if merge_all_data:
    # 1. Join/Merge in prevApps Data
    train_dataset = train_dataset.merge(datasets["previous_application_agg"], how='left', on='SK_ID_CURR')

    # 2. Join/Merge in Installments Payments  Data
    train_dataset = train_dataset.merge(datasets["installments_payments_agg"], how='left', on="SK_ID_CURR")

    # 3. Join/Merge in Credit Card Balance Data
    train_dataset = train_dataset.merge(datasets["credit_card_balance_agg"], how='left', on="SK_ID_CURR")
```

In [35]:
```
datasets["undersampled_application_train_4"] = train_dataset
```

In [36]:
```
train_dataset.shape
```

Out[36]:
```
(99825, 125)
```

In [37]:
```
train_dataset = datasets["undersampled_application_train_2"]
train_dataset = train_dataset.merge(datasets["previous_application_agg"], how='left', on='SK_ID_CURR')
train_dataset = train_dataset.merge(datasets["installments_payments_agg"], how='left', on="SK_ID_CURR")
train_dataset = train_dataset.merge(datasets["credit_card_balance_agg"], how='left', on="SK_ID_CURR")
train_dataset = train_dataset.drop(columns = 'weight')
datasets["undersampled_application_train_4_2"] = train_dataset
```

In [38]:
```
train_dataset.shape
```

Out[38]:
```
(49650, 125)
```

In [39]:
```
train_dataset.to_csv('train_dataset.csv', index=False)
```

In [40]:
```
X_kaggle_test= datasets["application_test"]

# merge primary table and secondary tables using features based on meta data and  aggregage stats
if merge_all_data:
    # 1. Join/Merge in prevApps Data
    X_kaggle_test = X_kaggle_test.merge(datasets["previous_application_agg"], how='left', on='SK_ID_CURR')

    # 2. Join/Merge in Installments Payments  Data
    X_kaggle_test = X_kaggle_test.merge(datasets["installments_payments_agg"], how='left', on="SK_ID_CURR")

    # 3. Join/Merge in Credit Card Balance Data
    X_kaggle_test = X_kaggle_test.merge(datasets["credit_card_balance_agg"], how='left', on="SK_ID_CURR")
```

In [41]:
```
X_kaggle_test.shape
```

Out[41]:
```
(48744, 124)
```

In [42]:
```
X_kaggle_test.to_csv('X_kaggle_test.csv', index=False)
```

From Phase 3

In the previous phase, I conducted feature engineering and obtained a dataset that I will be using in the current phase. I have also carried forward the feature dictionary obtained after hyperparameter tuning of the XGBoost model in the previous phase. Therefore, in this phase, I will be utilizing the same dataset and feature dictionary to perform further analysis.The train_dataset.csv file used in this phase of the project is derived from the training dataset in Phase 3. It is a CSV file that contains the merged undersampled data from various tables, including application train, previous application, installment payments, and credit card balance. Additionally, the file includes other engineered features created in the feature engineering section of Phase 3.

Loading Datasets and Constructing Pipeline

In [70]:
```
train_dataset = pd.read_csv("train_dataset.csv")
train_dataset.head()
```

Out[70]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_
0	100002	1	Cash loans	M	N	Y	0	2
1	100031	1	Cash loans	F	N	Y	0	1
2	100047	1	Cash loans	M	N	Y	0	2
3	100049	1	Cash loans	F	N	N	0	1
4	100096	1	Cash loans	F	N	Y	0	

5 rows × 125 columns

In [71]:
```
train_dataset.shape
```

Out[71]:
```
(49650, 125)
```

In [ ]:
```
# import pandas as pd
# import pandas_profiling

# # Create the report
# train_dataset_profile = pandas_profiling.ProfileReport(train_dataset)
# train_dataset_profile
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
```

The X_kaggle_test.csv is also created in Phase 3 of this project and contains the test data merged with other created features.

In [46]:
```
#train_dataset = pd.read_csv("train_dataset.csv")
X_kaggle_test = pd.read_csv("X_kaggle_test.csv")
```

```
X_kaggle_test.head()
```

Out[46]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	A
0	100001	Cash loans	F	N	Y	0	135000.0	
1	100005	Cash loans	M	N	Y	0	99000.0	
2	100013	Cash loans	M	Y	Y	0	202500.0	
3	100028	Cash loans	F	N	Y	2	315000.0	
4	100038	Cash loans	M	Y	N	1	180000.0	

5 rows × 124 columns

In [47]:
```
X_kaggle_test.shape
```

Out[47]:
```
(48744, 124)
```

In [48]:
```
# class to select numerical or categorical columns
class DataFrameCreation(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values

def pct(x):
    return round(100*x,3)

def get_pipeline(dataset, num_cols = None):

    numerical_features = []
    categorical_features = []
    for x in dataset:
        if(dataset[x].dtype == np.float64 or dataset[x].dtype == np.int64):
            numerical_features.append(x)
        else:
            categorical_features.append(x)
    numerical_features.remove('TARGET')
    numerical_features.remove('SK_ID_CURR')

    categorical_pipeline = Pipeline([
            ('selector', DataFrameCreation(categorical_features)),
            ('imputer', SimpleImputer(strategy='most_frequent')),
            ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
        ])

    # If columns are provided, we use only pass those columns to the model
    if num_cols == None:
        final_numerical_features = numerical_features
    else:
        final_numerical_features = num_cols

    numerical_pipeline = Pipeline([
            ('selector', DataFrameCreation(final_numerical_features)),
            ('imputer', SimpleImputer(strategy='mean')),
            ('std_scaler', StandardScaler()),
        ])

    data_pipeline = FeatureUnion(transformer_list=[
            ("numerical_pipeline", numerical_pipeline),
            ("categorical_pipeline", categorical_pipeline),
        ])

    selected_features = final_numerical_features + categorical_features + ["SK_ID_CURR"]
    tot_features = f"{len(selected_features)}:   Num:{len(final_numerical_features)},    Cat:{len(categorical_f

    print('Total Features:', tot_features)

    return data_pipeline, selected_features
```

In [49]:
```
data_pipeline, selected_features = get_pipeline(train_dataset)
```

```
Total Features: 124:   Num:107,    Cat:16
```

In [50]:
```
y_train = train_dataset['TARGET']
X_train = train_dataset[selected_features]
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

print(f"X train           shape: {X_train.shape}")
print(f"X test            shape: {X_test.shape}")
```

```
X train           shape: (39720, 124)
X test            shape: (9930, 124)
```

Checking the availability of GPU

Checking the availability of GPU

In [51]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cpu
```

Handling Missing values, stansrdizing the data using pipeline and Generating Tensors

In [52]:
```
# Handling missing values and standardizing the data
X_train_std = data_pipeline.fit_transform(X_train)
X_test_std = data_pipeline.transform(X_test)
X_kaggle_test_std = data_pipeline.transform(X_kaggle_test)

# Converting numpy arrays into float tensors using gpu device
X_train_tensor = torch.FloatTensor(X_train_std).to(device)
X_test_tensor = torch.FloatTensor(X_test_std).to(device)
X_kaggle_test_tensor = torch.FloatTensor(X_kaggle_test_std).to(device)

# Converting numpy arrays to float tensors and reshaping y_train and y_test
y_train_tensor = torch.FloatTensor(y_train.to_numpy()).to(device)
y_train_tensor = y_train_tensor.reshape(-1, 1)
y_test_tensor = torch.FloatTensor(y_test.to_numpy()).to(device)
y_test_tensor = y_test_tensor.reshape(-1, 1)
```

In [53]:
```
X_train_tensor.shape, X_test_tensor.shape, X_kaggle_test_tensor.shape
```

Out[53]:
```
(torch.Size([39720, 245]), torch.Size([9930, 245]), torch.Size([48744, 245]))
```

Using Selected Features from Phase3

In [54]:
```
# Loading features and importances from phase3
with open("features_dict_XG.pickle", 'rb') as handle:
    features_dict = pickle.load(handle)

# selecting features with importance values > 0
features = features_dict['features']
importances = features_dict['importances']
new_indices = [idx for idx, x in enumerate(importances) if x > 0]
new_importances = [x for idx, x in enumerate(importances) if x > 0]
new_features = [features[i] for i in new_indices]

# creating pipeline by joining numerical and categorical pipelines
num_attribs = new_features
data_pipeline, selected_features = get_pipeline(train_dataset, num_attribs)

# splitting the dataset into train and test datasets with selected features
y_train_sel, X_train_sel = train_dataset['TARGET'], train_dataset[selected_features]
X_kaggle_test_sel = X_kaggle_test[selected_features]
X_train_sel, X_test_sel, y_train_sel, y_test_sel = train_test_split(X_train_sel, y_train_sel, test_size=0.2, ra

# Handling missing values and standardizing the data using pipeline
X_train_sel_std, X_test_sel_std, X_kaggle_test_sel_std = data_pipeline.fit_transform(X_train_sel), data_pipelin

# Generating float tensors from numpy arrays using GPU device
X_train_sel_tensor, X_test_sel_tensor, X_kaggle_test_sel_tensor = map(lambda x: torch.FloatTensor(x).to(device)
y_train_sel_tensor, y_test_sel_tensor = map(lambda x: torch.FloatTensor(x.to_numpy()).reshape(-1, 1).to(device)

# Print the shapes of tensors
print(f"X train selected shape: {X_train_sel_tensor.shape}")
print(f"X test selected shape: {X_test_sel_tensor.shape}")
```

```
Total Features: 112:   Num:95,    Cat:16
X train selected shape: torch.Size([39720, 233])
X test selected shape: torch.Size([9930, 233])
```

In [55]:
```
%matplotlib inline
writer = SummaryWriter()
```

Evaluation metrics

The evaluation of submissions is conducted through the calculation of the area under the ROC curve, which measures the relationship between the predicted probability and the observed target. The SkLearn roc_auc_score function is utilized to compute the AUC or AUROC, effectively summarizing the information contained in the ROC curve into a single numerical value.

Submissions are evaluated on area under the ROC curve between the predicted probability and the observed target.

The SkLearn `roc_auc_score` function computes the area under the receiver operating characteristic (ROC) curve, which is also denoted by AUC or AUROC. By computing the area under the roc curve, the curve information is summarized in one number.

```
from sklearn.metrics import roc_auc_score
>>> y_true = np.array([0, 0, 1, 1])
>>> y_scores = np.array([0.1, 0.4, 0.35, 0.8])
```

```
>>> roc_auc_score(y_true, y_scores)
0.75
```

ACCURACY

It refers to the proportion of accurately classified data instances in relation to the overall number of data instances.

$$ \operatorname{Accuracy} = \frac{TN+TP}{TN+FP+TP+FN}\ $$

PRECISION:

Precision refers to the ratio of true positives to the sum of true positives and false positives.

$$ \operatorname{Precision} = \frac{TP}{TP+FP}\ $$

RECALL

It denotes the fraction of positive instances that are correctly identified as positive by the model. This metric is equivalent to the TPR (True Positive Rate).

$$ \operatorname{Recall} = \frac{TP}{TP+FN}\ $$

F1 SCORE

It is the harmonic mean of accuracy and recall, taking into account both false positives and false negatives. It is a useful metric for evaluating models on imbalanced datasets.

$$ \operatorname{F1Score} = \frac{Precision * Recall}{Precision + Recall}\ $$

AUC

The Area Under the Curve (AUC) metric is used to evaluate the performance of binary classification models by measuring the area under the Receiver Operating Characteristic (ROC) curve. It provides a single scalar value that represents the overall performance of the model across all possible classification thresholds. AUC is a widely used metric in machine learning because it is robust to class imbalance and insensitive to the specific classification threshold used. Higher values of AUC indicate better model performance.

In [56]:
```
try:
    expLog
except NameError:
    expLog = pd.DataFrame(columns=["exp_name", "learning_rate", "epochs",
                                   "Train Time (sec)",
                                   "Test Time (sec)",
                                   "Train Acc",
                                   "Test Acc",
                                   "Train AUC",
                                   "Test AUC",
                                   "Train F1",
                                   "Test F1"
                                  ])
```

Loss Function Used

The binary cross-entropy loss function will be utilized by this MLP class.

$$ CXE = -\frac{1}{m}\sum \limits_{i=1}^m (y_i \cdot log(p_i) + (1-y_i)\cdot log(1-p_i)) $$

Modules for Training and Testing

In [39]:
```
from sklearn.metrics import f1_score

def get_results(expLog, exp_name, learning_rate, epochs, model, train_time, test_time, X_train, y_train, X_test

    def test_metrics(X, y, model):
      X = X.to(device) # Move the input tensor to the GPU
      model.eval()
      with torch.no_grad():
          y_prob = model(X)
          y_pred = y_prob.cpu().detach().numpy().round()
          roc_auc = roc_auc_score(y, y_pred)
          accuracy = accuracy_score(y, y_pred)
          f1 = f1_score(y, y_pred)
      return accuracy, roc_auc, f1
```

```
    # Getting the results
    accuracy_train, roc_auc_train, f1_train = test_metrics(X_train, y_train, model)
    accuracy_test, roc_auc_test, f1_test = test_metrics(X_test, y_test, model)

    expLog.loc[len(expLog)] = [f"{exp_name}"] + list(np.round(
               [learning_rate, epochs, train_time, test_time,
                accuracy_train, accuracy_test, roc_auc_train, roc_auc_test, f1_train, f1_test],
    4))
    return expLog
```

In [40]:
```
from sklearn.metrics import f1_score

def train_and_test(X_train_tensor, y_train_tensor, X_test_tensor, y_test_tensor, model, optimizer, writer, lear
    # Move tensors to the GPU
    X_train_tensor = X_train_tensor.to(device)
    y_train_tensor = y_train_tensor.to(device)
    X_test_tensor = X_test_tensor.to(device)

    # Model to be trained on GPU
    model = model.to(device)

    print('Model Architecture:')
    print(model, '\n')

    print('Training the model:')
    model.train()

    for epoch_id in range(epochs):
        y_prob = model(X_train_tensor)
        loss = binary_cross_entropy(y_prob, y_train_tensor)
        writer.add_scalar("Train Loss", loss, epoch_id+1)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if epoch_id % 50 == 49:
            print(f"Epoch {epoch_id + 1}:")
            show_metrics(y_train_tensor, y_prob, epoch_id+1, writer)

    writer.flush()
    writer.close()
    print()

    # Testing the model
    model.eval()
    with torch.no_grad():
        y_test_pred_prob = model(X_test_tensor)
        y_test_tensor = y_test_tensor.to(device)
        print('Test data:')
        show_metrics(y_test_tensor, y_test_pred_prob, writer=None)

def show_metrics(y_true, y_prob, idx=0, writer=None):
    y_pred = y_prob.cpu().detach().numpy().round()

    # Move tensors to the CPU
    y_true = y_true.cpu()

    # Calculating metrics from actual and predicted values
    roc_auc = roc_auc_score(y_true, y_pred)
    accuracy = accuracy_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    if writer:
        # Adding info to tensorboard
        writer.add_scalar("Train ROC_AUC", roc_auc, idx)
        writer.add_scalar("Train Accuracy", accuracy, idx)
        writer.add_scalar("Train F1", f1, idx)

    # Printing accuracy, ROC_AUC, and F1 for reference
    print(f'Accuracy : {round(accuracy,4)} ; ROC_AUC : {round(roc_auc, 4)} ; F1 : {round(f1, 4)}')
```

In [41]:
```
def train_and_test(X_train_tensor, y_train_tensor, X_test_tensor, y_test_tensor, model, optimizer, writer, lear
    # Move tensors to the GPU
    X_train_tensor = X_train_tensor.to(device)
    y_train_tensor = y_train_tensor.to(device)
    X_test_tensor = X_test_tensor.to(device)
    y_test_tensor = y_test_tensor.to(device)

    # Model to be trained on GPU
    model = model.to(device)

    print('Model Architecture:')
    print(model, '\n')

    print('Training the model:')
    model.train()
```

```
    for epoch_id in range(epochs):
        y_prob = model(X_train_tensor)
        loss = binary_cross_entropy(y_prob, y_train_tensor)
        writer.add_scalar("Train Loss", loss, epoch_id+1)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if epoch_id % 50 == 49:
            print(f"Epoch {epoch_id + 1}:")
            show_metrics(y_train_tensor, y_prob, epoch_id+1, writer)

    writer.flush()
    writer.close()
    print()

    # Testing the model
    model.eval()
    with torch.no_grad():
        y_test_pred_prob = model(X_test_tensor)
        print('Test data:')
        show_metrics(y_test_tensor, y_test_pred_prob, writer=None)

def show_metrics(y_true, y_prob, idx=0, writer=None):
    y_pred = y_prob.cpu().detach().numpy().round()

    # Move tensors to the CPU
    y_true = y_true.cpu()

    # Calculating metrics from actual and predicted values
    roc_auc = roc_auc_score(y_true.cpu().numpy(), y_pred)
    accuracy = accuracy_score(y_true.cpu().numpy(), y_pred)
    f1 = f1_score(y_true.cpu().numpy(), y_pred)

    if writer:
        # Adding info to tensorboard
        writer.add_scalar("Train ROC_AUC", roc_auc, idx)
        writer.add_scalar("Train Accuracy", accuracy, idx)
        writer.add_scalar("Train F1", f1, idx)

    # Printing accuracy, ROC_AUC, and F1 for reference
    print(f'Accuracy : {round(accuracy,4)} ; ROC_AUC : {round(roc_auc, 4)} ; F1 : {round(f1, 4)}')
```

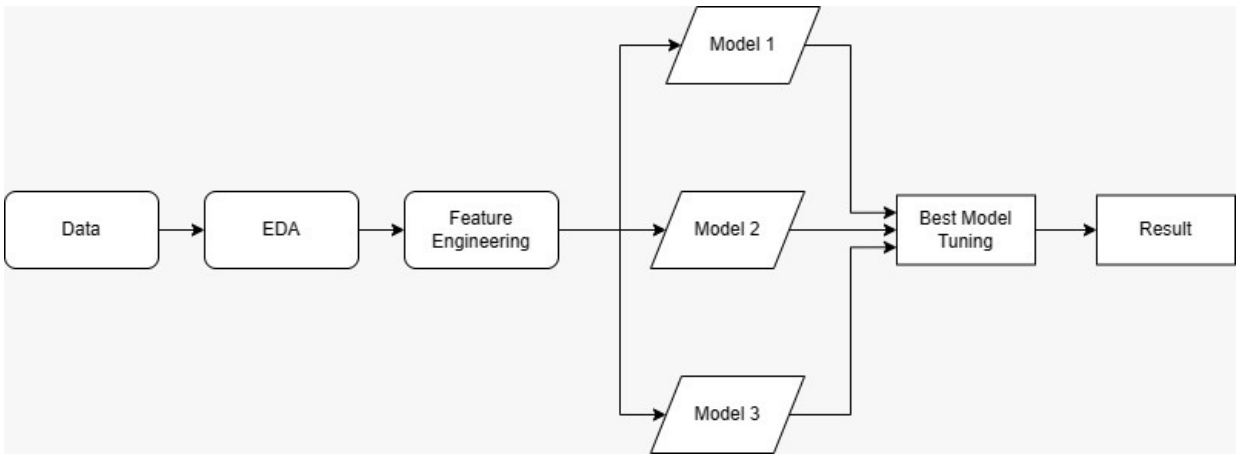
Model Pipeline

We will take HCDR data, preprocess it and apply feature engineering techniques as we did in phase 3. Then, after feature engineering, we will use the same feature selection method as we did in phase 3, where we will use the same feature dictionary. Next, we will develop three MLP models with varying depth and complexity. After this, we will select the best-performing model and perform hyperparameter tuning. We will compile all the results and analyze them, and then we will choose the best model based on its F1 and AUC scores. Finally, we will submit it as a Kaggle submission.

Block diagram of pipeline

In [66]:
```
from IPython.display import Image
Image(filename='p4block.jpeg')
```

Out[66]:



Model- 1 SIMPLE MLP

This is a simple neural network model built using PyTorch, a popular deep learning framework. The model architecture consists of a

single layer with a linear transformation followed by a sigmoid activation function. The input and output dimensions are defined based on the shape of the training data. The input dimension is set to the number of columns in the training data, and the output dimension is set to 1, which is appropriate for a binary classification problem

Experiment1: All features before feature selection

In [42]:
```
import torch
import torch.nn as nn

# Define input and output dimensions
dim_input = X_train_tensor.shape[1]
dim_output = 1

# Define the model architecture
model1 = torch.nn.Sequential(
    torch.nn.Linear(dim_input, dim_output),
    nn.Sigmoid()
)
```

In [43]:
```
from torchsummary import summary

# Print summary of model architecture
summary(model1, input_size=(X_train_tensor.shape[1],), device='cpu')
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                    [-1, 1]             246
           Sigmoid-2                    [-1, 1]               0
================================================================
Total params: 246
Trainable params: 246
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.00
----------------------------------------------------------------
```

In [45]:
```
import time
import numpy as np
from torch.optim import Adam

model = model1
learning_rate = 0.01
epochs = 1000
optimizer = Adam(model.parameters(), learning_rate)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.float32)
y_test=y_test_tensor
# Training the model
start_time = time.time()
train_and_test(X_train_tensor, y_train_tensor, X_test_tensor, y_test, model, optimizer, writer, learning_rate,
train_time = np.round(time.time() - start_time, 4)

# Testing the model
start_time = time.time()
train_and_test(X_train_tensor, y_train_tensor, X_test_tensor, y_test, model, optimizer, writer, learning_rate,
test_time = np.round(time.time() - start_time, 4)

print(f'Training time: {train_time} seconds')
print(f'Testing time: {test_time} seconds')
```

```
Model Architecture:
Sequential(
  (0): Linear(in_features=245, out_features=1, bias=True)
  (1): Sigmoid()
)

Training the model:
Epoch 50:
Accuracy : 0.687 ; ROC_AUC : 0.687 ; F1 : 0.6863
Epoch 100:
Accuracy : 0.6886 ; ROC_AUC : 0.6886 ; F1 : 0.6878
Epoch 150:
Accuracy : 0.6887 ; ROC_AUC : 0.6887 ; F1 : 0.6878
Epoch 200:
Accuracy : 0.6894 ; ROC_AUC : 0.6894 ; F1 : 0.6884
Epoch 250:
Accuracy : 0.6897 ; ROC_AUC : 0.6897 ; F1 : 0.6888
Epoch 300:
Accuracy : 0.6899 ; ROC_AUC : 0.6899 ; F1 : 0.689
Epoch 350:
Accuracy : 0.69 ; ROC_AUC : 0.69 ; F1 : 0.6891
Epoch 400:
Accuracy : 0.6898 ; ROC_AUC : 0.6898 ; F1 : 0.6889
```

```
Epoch 450:
Accuracy : 0.6901 ; ROC_AUC : 0.6901 ; F1 : 0.6893
Epoch 500:
Accuracy : 0.6899 ; ROC_AUC : 0.6899 ; F1 : 0.6891
Epoch 550:
Accuracy : 0.6901 ; ROC_AUC : 0.6901 ; F1 : 0.6893
Epoch 600:
Accuracy : 0.6902 ; ROC_AUC : 0.6902 ; F1 : 0.6894
Epoch 650:
Accuracy : 0.6905 ; ROC_AUC : 0.6905 ; F1 : 0.6897
Epoch 700:
Accuracy : 0.6906 ; ROC_AUC : 0.6906 ; F1 : 0.6898
Epoch 750:
Accuracy : 0.6906 ; ROC_AUC : 0.6906 ; F1 : 0.6897
Epoch 800:
Accuracy : 0.6907 ; ROC_AUC : 0.6907 ; F1 : 0.6898
Epoch 850:
Accuracy : 0.6907 ; ROC_AUC : 0.6907 ; F1 : 0.6898
Epoch 900:
Accuracy : 0.6906 ; ROC_AUC : 0.6906 ; F1 : 0.6898
Epoch 950:
Accuracy : 0.6907 ; ROC_AUC : 0.6907 ; F1 : 0.6899
Epoch 1000:
Accuracy : 0.6908 ; ROC_AUC : 0.6908 ; F1 : 0.69

Test data:
Accuracy : 0.6813 ; ROC_AUC : 0.6813 ; F1 : 0.6814
Model Architecture:
Sequential(
  (0): Linear(in_features=245, out_features=1, bias=True)
  (1): Sigmoid()
)

Training the model:
Epoch 50:
Accuracy : 0.6908 ; ROC_AUC : 0.6908 ; F1 : 0.69
Epoch 100:
Accuracy : 0.6908 ; ROC_AUC : 0.6908 ; F1 : 0.69
Epoch 150:
Accuracy : 0.6909 ; ROC_AUC : 0.6909 ; F1 : 0.6901
Epoch 200:
Accuracy : 0.6909 ; ROC_AUC : 0.6909 ; F1 : 0.6902
Epoch 250:
Accuracy : 0.691 ; ROC_AUC : 0.691 ; F1 : 0.6902
Epoch 300:
Accuracy : 0.6909 ; ROC_AUC : 0.6909 ; F1 : 0.6902
Epoch 350:
Accuracy : 0.691 ; ROC_AUC : 0.691 ; F1 : 0.6904
Epoch 400:
Accuracy : 0.691 ; ROC_AUC : 0.691 ; F1 : 0.6903
Epoch 450:
Accuracy : 0.6912 ; ROC_AUC : 0.6912 ; F1 : 0.6905
Epoch 500:
Accuracy : 0.6911 ; ROC_AUC : 0.6911 ; F1 : 0.6905
Epoch 550:
Accuracy : 0.6911 ; ROC_AUC : 0.6911 ; F1 : 0.6905
Epoch 600:
Accuracy : 0.691 ; ROC_AUC : 0.691 ; F1 : 0.6904
Epoch 650:
Accuracy : 0.6902 ; ROC_AUC : 0.6902 ; F1 : 0.6888
Epoch 700:
Accuracy : 0.6912 ; ROC_AUC : 0.6912 ; F1 : 0.6907
Epoch 750:
Accuracy : 0.6911 ; ROC_AUC : 0.6911 ; F1 : 0.6905
Epoch 800:
Accuracy : 0.691 ; ROC_AUC : 0.691 ; F1 : 0.6904
Epoch 850:
Accuracy : 0.6908 ; ROC_AUC : 0.6908 ; F1 : 0.6902
Epoch 900:
Accuracy : 0.6901 ; ROC_AUC : 0.6901 ; F1 : 0.6882
Epoch 950:
Accuracy : 0.6909 ; ROC_AUC : 0.6909 ; F1 : 0.6904
Epoch 1000:
Accuracy : 0.691 ; ROC_AUC : 0.691 ; F1 : 0.6904

Test data:
Accuracy : 0.6828 ; ROC_AUC : 0.6828 ; F1 : 0.6832
Training time: 5.0025 seconds
Testing time: 3.6912 seconds
```

In [48]:

```
exp_name = f"Model1 All"
expLog = get_results(expLog, exp_name, learning_rate, epochs, model, train_time, test_time, X_train_tensor, y_t
expLog
```

Out[48]:

	exp_name	learning_rate	epochs	Train Time (sec)	Test Time (sec)	Train Acc	Test Acc	Train AUC	Test AUC	Train F1	Test F1
0	Model1 All	0.01	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
1	Model1 All	0.01	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
2	Model1 All	0.01	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832

In [49]:
```
%load_ext tensorboard
```

In [50]:
```
tensorboard --logdir=runs
```

Experiment2: Selected features after x>0 from findings we did in phase 3

In [51]:
```
dim_input = X_train_sel_tensor.shape[1]
dim_output = 1
model1 = torch.nn.Sequential(
    torch.nn.Linear(dim_input, dim_output),
    nn.Sigmoid())
```

In [52]:
```
model = model1
learning_rate = 0.01
epochs = 1000
optimizer = Adam(model.parameters(), learning_rate)

y_test_tensor = torch.tensor(y_test_sel.values, dtype=torch.float32)
y_test_sel=y_test_tensor

# Training the model
start_time = time.time()
train_and_test(X_train_sel_tensor, y_train_sel_tensor, X_test_sel_tensor, y_test_sel, model, optimizer, writer,
train_time = np.round(time.time() - start_time, 4)

# Testing the model
start_time = time.time()
train_and_test(X_train_sel_tensor, y_train_sel_tensor, X_test_sel_tensor, y_test_sel, model, optimizer, writer,
test_time = np.round(time.time() - start_time, 4)

print(f'Training time: {train_time} seconds')
print(f'Testing time: {test_time} seconds')
```

```
Model Architecture:
Sequential(
  (0): Linear(in_features=233, out_features=1, bias=True)
  (1): Sigmoid()
)

Training the model:
Epoch 50:
Accuracy : 0.6873 ; ROC_AUC : 0.6873 ; F1 : 0.6872
Epoch 100:
Accuracy : 0.6883 ; ROC_AUC : 0.6883 ; F1 : 0.6875
Epoch 150:
Accuracy : 0.6885 ; ROC_AUC : 0.6885 ; F1 : 0.6876
Epoch 200:
Accuracy : 0.6895 ; ROC_AUC : 0.6895 ; F1 : 0.6886
Epoch 250:
Accuracy : 0.6897 ; ROC_AUC : 0.6897 ; F1 : 0.6888
Epoch 300:
Accuracy : 0.6896 ; ROC_AUC : 0.6896 ; F1 : 0.6888
Epoch 350:
Accuracy : 0.6898 ; ROC_AUC : 0.6898 ; F1 : 0.6889
Epoch 400:
Accuracy : 0.6897 ; ROC_AUC : 0.6897 ; F1 : 0.6888
Epoch 450:
Accuracy : 0.6896 ; ROC_AUC : 0.6896 ; F1 : 0.6886
Epoch 500:
Accuracy : 0.6898 ; ROC_AUC : 0.6898 ; F1 : 0.6889
Epoch 550:
Accuracy : 0.6897 ; ROC_AUC : 0.6897 ; F1 : 0.6888
Epoch 600:
Accuracy : 0.6899 ; ROC_AUC : 0.6899 ; F1 : 0.6892
Epoch 650:
Accuracy : 0.6899 ; ROC_AUC : 0.6899 ; F1 : 0.6891
Epoch 700:
Accuracy : 0.6899 ; ROC_AUC : 0.6899 ; F1 : 0.6892
Epoch 750:
Accuracy : 0.69 ; ROC_AUC : 0.69 ; F1 : 0.6892
Epoch 800:
Accuracy : 0.6899 ; ROC_AUC : 0.6899 ; F1 : 0.6891
Epoch 850:
Accuracy : 0.6899 ; ROC_AUC : 0.6899 ; F1 : 0.6891
Epoch 900:
Accuracy : 0.69 ; ROC_AUC : 0.69 ; F1 : 0.6896
```

```
Epoch 950:
Accuracy : 0.6897 ; ROC_AUC : 0.6897 ; F1 : 0.6889
Epoch 1000:
Accuracy : 0.69 ; ROC_AUC : 0.69 ; F1 : 0.6892

Test data:
Accuracy : 0.6812 ; ROC_AUC : 0.6812 ; F1 : 0.6813
Model Architecture:
Sequential(
  (0): Linear(in_features=233, out_features=1, bias=True)
  (1): Sigmoid()
)

Training the model:
Epoch 50:
Accuracy : 0.6899 ; ROC_AUC : 0.6899 ; F1 : 0.6892
Epoch 100:
Accuracy : 0.6899 ; ROC_AUC : 0.6899 ; F1 : 0.6892
Epoch 150:
Accuracy : 0.69 ; ROC_AUC : 0.69 ; F1 : 0.6893
Epoch 200:
Accuracy : 0.69 ; ROC_AUC : 0.69 ; F1 : 0.6896
Epoch 250:
Accuracy : 0.69 ; ROC_AUC : 0.69 ; F1 : 0.6892
Epoch 300:
Accuracy : 0.6899 ; ROC_AUC : 0.6899 ; F1 : 0.6892
Epoch 350:
Accuracy : 0.6899 ; ROC_AUC : 0.6899 ; F1 : 0.6893
Epoch 400:
Accuracy : 0.69 ; ROC_AUC : 0.69 ; F1 : 0.6894
Epoch 450:
Accuracy : 0.6905 ; ROC_AUC : 0.6905 ; F1 : 0.6912
Epoch 500:
Accuracy : 0.6902 ; ROC_AUC : 0.6902 ; F1 : 0.6896
Epoch 550:
Accuracy : 0.69 ; ROC_AUC : 0.69 ; F1 : 0.6893
Epoch 600:
Accuracy : 0.69 ; ROC_AUC : 0.69 ; F1 : 0.6894
Epoch 650:
Accuracy : 0.69 ; ROC_AUC : 0.69 ; F1 : 0.6894
Epoch 700:
Accuracy : 0.6906 ; ROC_AUC : 0.6906 ; F1 : 0.691
Epoch 750:
Accuracy : 0.6901 ; ROC_AUC : 0.6901 ; F1 : 0.6892
Epoch 800:
Accuracy : 0.6902 ; ROC_AUC : 0.6902 ; F1 : 0.6896
Epoch 850:
Accuracy : 0.6901 ; ROC_AUC : 0.6901 ; F1 : 0.6895
Epoch 900:
Accuracy : 0.6902 ; ROC_AUC : 0.6902 ; F1 : 0.6896
Epoch 950:
Accuracy : 0.6903 ; ROC_AUC : 0.6903 ; F1 : 0.6897
Epoch 1000:
Accuracy : 0.6905 ; ROC_AUC : 0.6905 ; F1 : 0.6904

Test data:
Accuracy : 0.6814 ; ROC_AUC : 0.6814 ; F1 : 0.6816
Training time: 2.297 seconds
Testing time: 2.2334 seconds
```

In [53]:
```
exp_name = f"Model1 selected"
expLog = get_results(expLog, exp_name, learning_rate, epochs, model, train_time, test_time, X_train_sel_tensor,
expLog
```

Out[53]:

	exp_name	learning_rate	epochs	Train Time (sec)	Test Time (sec)	Train Acc	Test Acc	Train AUC	Test AUC	Train F1	Test F1
0	Model1 All	0.01	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
1	Model1 All	0.01	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
2	Model1 All	0.01	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
3	Model1 selected	0.01	1000.0	2.2970	2.2334	0.6902	0.6814	0.6902	0.6814	0.6896	0.6816

In [54]:
```
%reload_ext tensorboard
```

In [55]:
```
tensorboard --logdir=runs
```

```
Reusing TensorBoard on port 6006 (pid 4280), started 0:00:10 ago. (Use '!kill 4280' to kill it.)
```

Model-2

Model 2 is a PyTorch implementation of a Multi-Layer Perceptron (MLP) with batch normalization and dropout regularization to reduce overfitting. The MLP consists of 6 hidden layers with 512, 256, 128, 64, 32, and 1 neurons respectively. The input size is specified when the model is initialized. The activation function used is the rectified linear unit (ReLU) for the hidden layers and the sigmoid function for

the output layer. The dropout rate is set to 0.5, which means that 50% of the neurons in the hidden layers will be randomly deactivated during training to prevent overfitting.

Experiment1: All Features

In [56]:
```
import torch.nn as nn

class EnhancedMLP(nn.Module):
    def __init__(self, input_size):
        super(EnhancedMLP, self).__init__()
        self.hl1 = nn.Linear(input_size, 512)
        self.bn1 = nn.BatchNorm1d(512)
        self.hl2 = nn.Linear(512, 256)
        self.bn2 = nn.BatchNorm1d(256)
        self.hl3 = nn.Linear(256, 128)
        self.bn3 = nn.BatchNorm1d(128)
        self.hl4 = nn.Linear(128, 64)
        self.bn4 = nn.BatchNorm1d(64)
        self.hl5 = nn.Linear(64, 32)
        self.bn5 = nn.BatchNorm1d(32)
        self.hl6 = nn.Linear(32, 1)
        self.activation = nn.ReLU()
        self.sigmoid = nn.Sigmoid()
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        x = self.activation(self.bn1(self.hl1(x)))
        x = self.dropout(x)
        x = self.activation(self.bn2(self.hl2(x)))
        x = self.dropout(x)
        x = self.activation(self.bn3(self.hl3(x)))
        x = self.dropout(x)
        x = self.activation(self.bn4(self.hl4(x)))
        x = self.dropout(x)
        x = self.activation(self.bn5(self.hl5(x)))
        x = self.sigmoid(self.hl6(x))
        return x

model2 = EnhancedMLP(X_train_tensor.shape[1])
```

In [57]:
```
from torchsummary import summary

# Print summary of model architecture
summary(model2, input_size=(X_train_tensor.shape[1],), device='cpu')
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 512]         125,952
       BatchNorm1d-2                  [-1, 512]           1,024
              ReLU-3                  [-1, 512]               0
           Dropout-4                  [-1, 512]               0
            Linear-5                  [-1, 256]         131,328
       BatchNorm1d-6                  [-1, 256]             512
              ReLU-7                  [-1, 256]               0
           Dropout-8                  [-1, 256]               0
            Linear-9                  [-1, 128]          32,896
      BatchNorm1d-10                  [-1, 128]             256
             ReLU-11                  [-1, 128]               0
          Dropout-12                  [-1, 128]               0
           Linear-13                   [-1, 64]           8,256
      BatchNorm1d-14                   [-1, 64]             128
             ReLU-15                   [-1, 64]               0
          Dropout-16                   [-1, 64]               0
           Linear-17                   [-1, 32]           2,080
      BatchNorm1d-18                   [-1, 32]              64
             ReLU-19                   [-1, 32]               0
           Linear-20                    [-1, 1]              33
          Sigmoid-21                    [-1, 1]               0
================================================================
Total params: 302,529
Trainable params: 302,529
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.03
Params size (MB): 1.15
Estimated Total Size (MB): 1.19
----------------------------------------------------------------
```

In [58]:
```
model = model2
learning_rate = 0.01
epochs = 1000
optimizer = Adam(model.parameters(), learning_rate)

# Training the model
```

```
start_time = time.time()
train_and_test(X_train_tensor, y_train_tensor, X_test_tensor, y_test, model, optimizer, writer, learning_rate,
train_time = np.round(time.time() - start_time, 4)

# Testing the model
start_time = time.time()
train_and_test(X_train_tensor, y_train_tensor, X_test_tensor, y_test, model, optimizer, writer, learning_rate,
test_time = np.round(time.time() - start_time, 4)

print(f'Training time: {train_time} seconds')
print(f'Testing time: {test_time} seconds')
```

```
Model Architecture:
EnhancedMLP(
  (hl1): Linear(in_features=245, out_features=512, bias=True)
  (bn1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl2): Linear(in_features=512, out_features=256, bias=True)
  (bn2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl3): Linear(in_features=256, out_features=128, bias=True)
  (bn3): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl4): Linear(in_features=128, out_features=64, bias=True)
  (bn4): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl5): Linear(in_features=64, out_features=32, bias=True)
  (bn5): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl6): Linear(in_features=32, out_features=1, bias=True)
  (activation): ReLU()
  (sigmoid): Sigmoid()
  (dropout): Dropout(p=0.5, inplace=False)
)

Training the model:
Epoch 50:
Accuracy : 0.7291 ; ROC_AUC : 0.7291 ; F1 : 0.7321
Epoch 100:
Accuracy : 0.7747 ; ROC_AUC : 0.7747 ; F1 : 0.7781
Epoch 150:
Accuracy : 0.8133 ; ROC_AUC : 0.8133 ; F1 : 0.8241
Epoch 200:
Accuracy : 0.8493 ; ROC_AUC : 0.8493 ; F1 : 0.8476
Epoch 250:
Accuracy : 0.8648 ; ROC_AUC : 0.8648 ; F1 : 0.8691
Epoch 300:
Accuracy : 0.881 ; ROC_AUC : 0.881 ; F1 : 0.878
Epoch 350:
Accuracy : 0.8864 ; ROC_AUC : 0.8864 ; F1 : 0.8859
Epoch 400:
Accuracy : 0.8921 ; ROC_AUC : 0.8921 ; F1 : 0.8936
Epoch 450:
Accuracy : 0.9012 ; ROC_AUC : 0.9012 ; F1 : 0.9009
Epoch 500:
Accuracy : 0.9074 ; ROC_AUC : 0.9074 ; F1 : 0.907
Epoch 550:
Accuracy : 0.9115 ; ROC_AUC : 0.9115 ; F1 : 0.9116
Epoch 600:
Accuracy : 0.912 ; ROC_AUC : 0.9119 ; F1 : 0.914
Epoch 650:
Accuracy : 0.9182 ; ROC_AUC : 0.9182 ; F1 : 0.9183
Epoch 700:
Accuracy : 0.9216 ; ROC_AUC : 0.9215 ; F1 : 0.9222
Epoch 750:
Accuracy : 0.9243 ; ROC_AUC : 0.9243 ; F1 : 0.9239
Epoch 800:
Accuracy : 0.9245 ; ROC_AUC : 0.9245 ; F1 : 0.9251
Epoch 850:
Accuracy : 0.9288 ; ROC_AUC : 0.9288 ; F1 : 0.9288
Epoch 900:
Accuracy : 0.9266 ; ROC_AUC : 0.9266 ; F1 : 0.9273
Epoch 950:
Accuracy : 0.9254 ; ROC_AUC : 0.9253 ; F1 : 0.9264
Epoch 1000:
Accuracy : 0.9292 ; ROC_AUC : 0.9292 ; F1 : 0.9293

Test data:
Accuracy : 0.638 ; ROC_AUC : 0.6382 ; F1 : 0.6726
Model Architecture:
EnhancedMLP(
  (hl1): Linear(in_features=245, out_features=512, bias=True)
  (bn1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl2): Linear(in_features=512, out_features=256, bias=True)
  (bn2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl3): Linear(in_features=256, out_features=128, bias=True)
  (bn3): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl4): Linear(in_features=128, out_features=64, bias=True)
  (bn4): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl5): Linear(in_features=64, out_features=32, bias=True)
  (bn5): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl6): Linear(in_features=32, out_features=1, bias=True)
  (activation): ReLU()
  (sigmoid): Sigmoid()
```

```
  (dropout): Dropout(p=0.5, inplace=False)
)

Training the model:
Epoch 50:
Accuracy : 0.9334 ; ROC_AUC : 0.9334 ; F1 : 0.9331
Epoch 100:
Accuracy : 0.9323 ; ROC_AUC : 0.9323 ; F1 : 0.932
Epoch 150:
Accuracy : 0.9337 ; ROC_AUC : 0.9337 ; F1 : 0.9342
Epoch 200:
Accuracy : 0.9333 ; ROC_AUC : 0.9333 ; F1 : 0.9336
Epoch 250:
Accuracy : 0.936 ; ROC_AUC : 0.936 ; F1 : 0.9358
Epoch 300:
Accuracy : 0.9383 ; ROC_AUC : 0.9383 ; F1 : 0.9382
Epoch 350:
Accuracy : 0.9364 ; ROC_AUC : 0.9364 ; F1 : 0.9363
Epoch 400:
Accuracy : 0.9371 ; ROC_AUC : 0.9371 ; F1 : 0.9372
Epoch 450:
Accuracy : 0.9402 ; ROC_AUC : 0.9402 ; F1 : 0.9403
Epoch 500:
Accuracy : 0.9397 ; ROC_AUC : 0.9397 ; F1 : 0.9396
Epoch 550:
Accuracy : 0.9365 ; ROC_AUC : 0.9365 ; F1 : 0.936
Epoch 600:
Accuracy : 0.9419 ; ROC_AUC : 0.9419 ; F1 : 0.9422
Epoch 650:
Accuracy : 0.9436 ; ROC_AUC : 0.9436 ; F1 : 0.9436
Epoch 700:
Accuracy : 0.9401 ; ROC_AUC : 0.9401 ; F1 : 0.9404
Epoch 750:
Accuracy : 0.9425 ; ROC_AUC : 0.9426 ; F1 : 0.9422
Epoch 800:
Accuracy : 0.9445 ; ROC_AUC : 0.9445 ; F1 : 0.9447
Epoch 850:
Accuracy : 0.9447 ; ROC_AUC : 0.9447 ; F1 : 0.945
Epoch 900:
Accuracy : 0.9458 ; ROC_AUC : 0.9458 ; F1 : 0.946
Epoch 950:
Accuracy : 0.9449 ; ROC_AUC : 0.9449 ; F1 : 0.945
Epoch 1000:
Accuracy : 0.945 ; ROC_AUC : 0.945 ; F1 : 0.9454

Test data:
Accuracy : 0.6346 ; ROC_AUC : 0.6349 ; F1 : 0.6661
Training time: 27.099 seconds
Testing time: 28.2518 seconds
```

In [59]:
```
exp_name = f"Model 2 Enhanced all "
expLog = get_results(expLog, exp_name, learning_rate, epochs, model, train_time, test_time, X_train_tensor, y_t
expLog
```

Out[59]:

	exp_name	learning_rate	epochs	Train Time (sec)	Test Time (sec)	Train Acc	Test Acc	Train AUC	Test AUC	Train F1	Test F1
0	Model1 All	0.01	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
1	Model1 All	0.01	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
2	Model1 All	0.01	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
3	Model1 selected	0.01	1000.0	2.2970	2.2334	0.6902	0.6814	0.6902	0.6814	0.6896	0.6816
4	Model 2 Enhanced all	0.01	1000.0	27.0990	28.2518	0.9990	0.6346	0.9990	0.6349	0.9990	0.6661

In [60]:
```
%reload_ext tensorboard
```

In [61]:
```
tensorboard --logdir=runs
```

```
Reusing TensorBoard on port 6006 (pid 4280), started 0:01:12 ago. (Use '!kill 4280' to kill it.)
```

Experiment2

To optimize the performance of the model, the learning rate and the number of epochs will be adjusted based on the findings from Experiment1.

In [62]:
```
model2 = EnhancedMLP(X_train_tensor.shape[1])
model = model2
learning_rate = 0.001
epochs = 50
optimizer = Adam(model.parameters(), learning_rate)

# Training the model
```

```
start_time = time.time()
train_and_test(X_train_tensor, y_train_tensor, X_test_tensor, y_test, model, optimizer, writer, learning_rate,
train_time = np.round(time.time() - start_time, 4)

# Testing the model
start_time = time.time()
train_and_test(X_train_tensor, y_train_tensor, X_test_tensor, y_test, model, optimizer, writer, learning_rate,
test_time = np.round(time.time() - start_time, 4)

print(f'Training time: {train_time} seconds')
print(f'Testing time: {test_time} seconds')

exp_name = f"Model 2 enhanced 2"
expLog = get_results(expLog, exp_name, learning_rate, epochs, model, train_time, test_time, X_train_tensor, y_t
expLog
```

```
Model Architecture:
EnhancedMLP(
  (hl1): Linear(in_features=245, out_features=512, bias=True)
  (bn1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl2): Linear(in_features=512, out_features=256, bias=True)
  (bn2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl3): Linear(in_features=256, out_features=128, bias=True)
  (bn3): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl4): Linear(in_features=128, out_features=64, bias=True)
  (bn4): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl5): Linear(in_features=64, out_features=32, bias=True)
  (bn5): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl6): Linear(in_features=32, out_features=1, bias=True)
  (activation): ReLU()
  (sigmoid): Sigmoid()
  (dropout): Dropout(p=0.5, inplace=False)
)

Training the model:
Epoch 50:
Accuracy : 0.698 ; ROC_AUC : 0.698 ; F1 : 0.6988

Test data:
Accuracy : 0.6811 ; ROC_AUC : 0.6811 ; F1 : 0.6796
Model Architecture:
EnhancedMLP(
  (hl1): Linear(in_features=245, out_features=512, bias=True)
  (bn1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl2): Linear(in_features=512, out_features=256, bias=True)
  (bn2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl3): Linear(in_features=256, out_features=128, bias=True)
  (bn3): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl4): Linear(in_features=128, out_features=64, bias=True)
  (bn4): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl5): Linear(in_features=64, out_features=32, bias=True)
  (bn5): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl6): Linear(in_features=32, out_features=1, bias=True)
  (activation): ReLU()
  (sigmoid): Sigmoid()
  (dropout): Dropout(p=0.5, inplace=False)
)

Training the model:
Epoch 50:
Accuracy : 0.7204 ; ROC_AUC : 0.7204 ; F1 : 0.7228

Test data:
Accuracy : 0.6806 ; ROC_AUC : 0.6807 ; F1 : 0.6925
Training time: 1.4786 seconds
Testing time: 1.407 seconds
```

Out[62]:

	exp_name	learning_rate	epochs	Train Time (sec)	Test Time (sec)	Train Acc	Test Acc	Train AUC	Test AUC	Train F1	Test F1
0	Model1 All	0.010	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
1	Model1 All	0.010	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
2	Model1 All	0.010	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
3	Model1 selected	0.010	1000.0	2.2970	2.2334	0.6902	0.6814	0.6902	0.6814	0.6896	0.6816
4	Model 2 Enhanced all	0.010	1000.0	27.0990	28.2518	0.9990	0.6346	0.9990	0.6349	0.9990	0.6661
5	Model 2 enhanced 2	0.001	50.0	1.4786	1.4070	0.7411	0.6806	0.7411	0.6807	0.7501	0.6925

In [63]:
```
%reload_ext tensorboard
```

In [64]:
```
tensorboard --logdir=runs
```

```
Reusing TensorBoard on port 6006 (pid 4280), started 0:01:28 ago. (Use '!kill 4280' to kill it.)
```

Experiment2: Selected features after p>0 from findings we did in phase 2

Experiment3: Selected features after x>0 from findings we did in phase 3

In [65]:
```
model2 = EnhancedMLP(X_train_sel_tensor.shape[1])
model = model2
learning_rate = 0.001
epochs = 50
optimizer = Adam(model.parameters(), learning_rate)

 #Training the model
start_time = time.time()
train_and_test(X_train_sel_tensor, y_train_sel_tensor, X_test_sel_tensor, y_test_sel, model, optimizer, writer,
train_time = np.round(time.time() - start_time, 4)

# Testing the model
start_time = time.time()
train_and_test(X_train_sel_tensor, y_train_sel_tensor, X_test_sel_tensor, y_test_sel, model, optimizer, writer,
test_time = np.round(time.time() - start_time, 4)

print(f'Training time: {train_time} seconds')
print(f'Testing time: {test_time} seconds')


exp_name = f"Model 2 enhanced and selected "
expLog = get_results(expLog, exp_name, learning_rate, epochs, model, train_time, test_time, X_train_sel_tensor,
expLog
```

```
Model Architecture:
EnhancedMLP(
  (hl1): Linear(in_features=233, out_features=512, bias=True)
  (bn1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl2): Linear(in_features=512, out_features=256, bias=True)
  (bn2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl3): Linear(in_features=256, out_features=128, bias=True)
  (bn3): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl4): Linear(in_features=128, out_features=64, bias=True)
  (bn4): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl5): Linear(in_features=64, out_features=32, bias=True)
  (bn5): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl6): Linear(in_features=32, out_features=1, bias=True)
  (activation): ReLU()
  (sigmoid): Sigmoid()
  (dropout): Dropout(p=0.5, inplace=False)
)

Training the model:
Epoch 50:
Accuracy : 0.6971 ; ROC_AUC : 0.6971 ; F1 : 0.6968

Test data:
Accuracy : 0.6835 ; ROC_AUC : 0.6835 ; F1 : 0.6842
Model Architecture:
EnhancedMLP(
  (hl1): Linear(in_features=233, out_features=512, bias=True)
  (bn1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl2): Linear(in_features=512, out_features=256, bias=True)
  (bn2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl3): Linear(in_features=256, out_features=128, bias=True)
  (bn3): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl4): Linear(in_features=128, out_features=64, bias=True)
  (bn4): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl5): Linear(in_features=64, out_features=32, bias=True)
  (bn5): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl6): Linear(in_features=32, out_features=1, bias=True)
  (activation): ReLU()
  (sigmoid): Sigmoid()
  (dropout): Dropout(p=0.5, inplace=False)
)

Training the model:
Epoch 50:
Accuracy : 0.7176 ; ROC_AUC : 0.7176 ; F1 : 0.7177

Test data:
Accuracy : 0.6826 ; ROC_AUC : 0.6826 ; F1 : 0.6904
Training time: 1.4156 seconds
Testing time: 1.4354 seconds
```

Out[65]:

	exp_name	learning_rate	epochs	Train Time (sec)	Test Time (sec)	Train Acc	Test Acc	Train AUC	Test AUC	Train F1	Test F1
0	Model1 All	0.010	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
1	Model1 All	0.010	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
2	Model1 All	0.010	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
3	Model1 selected	0.010	1000.0	2.2970	2.2334	0.6902	0.6814	0.6902	0.6814	0.6896	0.6816
4	Model 2 Enhanced all	0.010	1000.0	27.0990	28.2518	0.9990	0.6346	0.9990	0.6349	0.9990	0.6661
5	Model 2 enhanced 2	0.001	50.0	1.4786	1.4070	0.7411	0.6806	0.7411	0.6807	0.7501	0.6925
6	Model 2 enhanced and selected	0.001	50.0	1.4156	1.4354	0.7364	0.6826	0.7364	0.6826	0.7413	0.6904

In [66]:
```
%reload_ext tensorboard
```

In [67]:
```
tensorboard --logdir=runs
```

```
Reusing TensorBoard on port 6006 (pid 4280), started 0:01:32 ago. (Use '!kill 4280' to kill it.)
```

Experiment4: Experiment3 by changing learning rate and epochs

In [68]:
```
model2 = EnhancedMLP(X_train_sel_tensor.shape[1])
model = model2
learning_rate = 0.0005
epochs = 50
optimizer = Adam(model.parameters(), learning_rate)

 #Training the model
start_time = time.time()
train_and_test(X_train_sel_tensor, y_train_sel_tensor, X_test_sel_tensor, y_test_sel, model, optimizer, writer,
train_time = np.round(time.time() - start_time, 4)

# Testing the model
start_time = time.time()
train_and_test(X_train_sel_tensor, y_train_sel_tensor, X_test_sel_tensor, y_test_sel, model, optimizer, writer,
test_time = np.round(time.time() - start_time, 4)

print(f'Training time: {train_time} seconds')
print(f'Testing time: {test_time} seconds')

exp_name = f"Model 3 change learning rate and epochs and selected "
expLog = get_results(expLog, exp_name, learning_rate, epochs, model, train_time, test_time, X_train_sel_tensor,
expLog
```

```
Model Architecture:
EnhancedMLP(
  (hl1): Linear(in_features=233, out_features=512, bias=True)
  (bn1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl2): Linear(in_features=512, out_features=256, bias=True)
  (bn2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl3): Linear(in_features=256, out_features=128, bias=True)
  (bn3): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl4): Linear(in_features=128, out_features=64, bias=True)
  (bn4): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl5): Linear(in_features=64, out_features=32, bias=True)
  (bn5): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl6): Linear(in_features=32, out_features=1, bias=True)
  (activation): ReLU()
  (sigmoid): Sigmoid()
  (dropout): Dropout(p=0.5, inplace=False)
)

Training the model:
Epoch 50:
Accuracy : 0.6871 ; ROC_AUC : 0.6871 ; F1 : 0.6853

Test data:
Accuracy : 0.6799 ; ROC_AUC : 0.6799 ; F1 : 0.6865
Model Architecture:
EnhancedMLP(
  (hl1): Linear(in_features=233, out_features=512, bias=True)
  (bn1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl2): Linear(in_features=512, out_features=256, bias=True)
  (bn2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl3): Linear(in_features=256, out_features=128, bias=True)
  (bn3): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl4): Linear(in_features=128, out_features=64, bias=True)
  (bn4): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl5): Linear(in_features=64, out_features=32, bias=True)
  (bn5): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl6): Linear(in_features=32, out_features=1, bias=True)
  (activation): ReLU()
  (sigmoid): Sigmoid()
  (dropout): Dropout(p=0.5, inplace=False)
)

Training the model:
Epoch 50:
Accuracy : 0.7015 ; ROC_AUC : 0.7015 ; F1 : 0.7036

Test data:
Accuracy : 0.6816 ; ROC_AUC : 0.6817 ; F1 : 0.6915
Training time: 1.4849 seconds
Testing time: 1.4059 seconds
```

Out[68]:

	exp_name	learning_rate	epochs	Train Time (sec)	Test Time (sec)	Train Acc	Test Acc	Train AUC	Test AUC	Train F1	Test F1
0	Model1 All	0.0100	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
1	Model1 All	0.0100	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
2	Model1 All	0.0100	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
3	Model1 selected	0.0100	1000.0	2.2970	2.2334	0.6902	0.6814	0.6902	0.6814	0.6896	0.6816
4	Model 2 Enhanced all	0.0100	1000.0	27.0990	28.2518	0.9990	0.6346	0.9990	0.6349	0.9990	0.6661
5	Model 2 enhanced 2	0.0010	50.0	1.4786	1.4070	0.7411	0.6806	0.7411	0.6807	0.7501	0.6925
6	Model 2 enhanced and selected	0.0010	50.0	1.4156	1.4354	0.7364	0.6826	0.7364	0.6826	0.7413	0.6904
7	Model 3 change learning rate and epochs and se...	0.0005	50.0	1.4849	1.4059	0.7101	0.6816	0.7101	0.6817	0.7165	0.6915

In [69]:
```
%reload_ext tensorboard
```

In [70]:
```
tensorboard --logdir=runs
```

```
Reusing TensorBoard on port 6006 (pid 4280), started 0:01:51 ago. (Use '!kill 4280' to kill it.)
```

Model3

Model 3 is PyTorch implementation of a Deep Wider MLP architecture. It is similar to the previous MLP implementation but with more layers and wider dimensions. This model consists of 8 hidden layers with 1024, 512, 256, 128, 64, 32, 16, and 1 neurons respectively. The input size is specified when the model is initialized. The activation function used is the rectified linear unit (ReLU) for the hidden layers and the sigmoid function for the output layer. The dropout rate is set to 0.5 to prevent overfitting. This model is capable of taking a tensor input and returning a tensor output with a single element.

The Architecture of the model which resulted in with the best accuracy and AUC score is 1024 -relu- 512-relu-256-relu-128-relu-63-relu-

32-relu-16-relu-1-signmoid

In [71]:
```python
# Deep Wider
import torch.nn as nn

class DeeperWiderMLP(nn.Module):
    def __init__(self, input_size):
        super(DeeperWiderMLP, self).__init__()
        self.hl1 = nn.Linear(input_size, 1024)
        self.bn1 = nn.BatchNorm1d(1024)
        self.hl2 = nn.Linear(1024, 512)
        self.bn2 = nn.BatchNorm1d(512)
        self.hl3 = nn.Linear(512, 256)
        self.bn3 = nn.BatchNorm1d(256)
        self.hl4 = nn.Linear(256, 128)
        self.bn4 = nn.BatchNorm1d(128)
        self.hl5 = nn.Linear(128, 64)
        self.bn5 = nn.BatchNorm1d(64)
        self.hl6 = nn.Linear(64, 32)
        self.bn6 = nn.BatchNorm1d(32)
        self.hl7 = nn.Linear(32, 16)
        self.bn7 = nn.BatchNorm1d(16)
        self.hl8 = nn.Linear(16, 1)
        self.activation = nn.ReLU()
        self.sigmoid = nn.Sigmoid()
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        x = self.activation(self.bn1(self.hl1(x)))
        x = self.dropout(x)
        x = self.activation(self.bn2(self.hl2(x)))
        x = self.dropout(x)
        x = self.activation(self.bn3(self.hl3(x)))
        x = self.dropout(x)
        x = self.activation(self.bn4(self.hl4(x)))
        x = self.dropout(x)
        x = self.activation(self.bn5(self.hl5(x)))
        x = self.dropout(x)
        x = self.activation(self.bn6(self.hl6(x)))
        x = self.dropout(x)
        x = self.activation(self.bn7(self.hl7(x)))
        x = self.sigmoid(self.hl8(x))
        return x
```

In [72]:
```python
from torchsummary import summary
model = DeeperWiderMLP(X_train_tensor.shape[1])
# Print summary of model architecture
summary(model, input_size=(X_train_tensor.shape[1],), device='cpu')
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1024]         251,904
       BatchNorm1d-2                [-1, 1024]           2,048
              ReLU-3                [-1, 1024]               0
           Dropout-4                [-1, 1024]               0
            Linear-5                 [-1, 512]         524,800
       BatchNorm1d-6                 [-1, 512]           1,024
              ReLU-7                 [-1, 512]               0
           Dropout-8                 [-1, 512]               0
            Linear-9                 [-1, 256]         131,328
      BatchNorm1d-10                 [-1, 256]             512
             ReLU-11                 [-1, 256]               0
          Dropout-12                 [-1, 256]               0
           Linear-13                 [-1, 128]          32,896
      BatchNorm1d-14                 [-1, 128]             256
             ReLU-15                 [-1, 128]               0
          Dropout-16                 [-1, 128]               0
           Linear-17                  [-1, 64]           8,256
      BatchNorm1d-18                  [-1, 64]             128
             ReLU-19                  [-1, 64]               0
          Dropout-20                  [-1, 64]               0
           Linear-21                  [-1, 32]           2,080
      BatchNorm1d-22                  [-1, 32]              64
             ReLU-23                  [-1, 32]               0
          Dropout-24                  [-1, 32]               0
           Linear-25                  [-1, 16]             528
      BatchNorm1d-26                  [-1, 16]              32
             ReLU-27                  [-1, 16]               0
           Linear-28                   [-1, 1]              17
          Sigmoid-29                   [-1, 1]               0
================================================================
Total params: 955,873
Trainable params: 955,873
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.06
Params size (MB): 3.65
Estimated Total Size (MB): 3.71
----------------------------------------------------------------
```

Experiment1 - All features

In [73]:

```python
model = DeeperWiderMLP(X_train_tensor.shape[1])
model = model
learning_rate = 0.001
epochs = 50
optimizer = Adam(model.parameters(), learning_rate)

# Training the model
start_time = time.time()
train_and_test(X_train_tensor, y_train_tensor, X_test_tensor, y_test, model, optimizer, writer, learning_rate,
train_time = np.round(time.time() - start_time, 4)

# Testing the model
start_time = time.time()
train_and_test(X_train_tensor, y_train_tensor, X_test_tensor, y_test, model, optimizer, writer, learning_rate,
test_time = np.round(time.time() - start_time, 4)

print(f'Training time: {train_time} seconds')
print(f'Testing time: {test_time} seconds')
```

```
Model Architecture:
DeeperWiderMLP(
  (hl1): Linear(in_features=245, out_features=1024, bias=True)
  (bn1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl2): Linear(in_features=1024, out_features=512, bias=True)
  (bn2): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl3): Linear(in_features=512, out_features=256, bias=True)
  (bn3): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl4): Linear(in_features=256, out_features=128, bias=True)
  (bn4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl5): Linear(in_features=128, out_features=64, bias=True)
  (bn5): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl6): Linear(in_features=64, out_features=32, bias=True)
  (bn6): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl7): Linear(in_features=32, out_features=16, bias=True)
  (bn7): BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl8): Linear(in_features=16, out_features=1, bias=True)
  (activation): ReLU()
  (sigmoid): Sigmoid()
  (dropout): Dropout(p=0.5, inplace=False)
)

Training the model:
Epoch 50:
Accuracy : 0.6977 ; ROC_AUC : 0.6977 ; F1 : 0.6927

Test data:
Accuracy : 0.6839 ; ROC_AUC : 0.6838 ; F1 : 0.6756
Model Architecture:
DeeperWiderMLP(
  (hl1): Linear(in_features=245, out_features=1024, bias=True)
  (bn1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl2): Linear(in_features=1024, out_features=512, bias=True)
  (bn2): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl3): Linear(in_features=512, out_features=256, bias=True)
  (bn3): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl4): Linear(in_features=256, out_features=128, bias=True)
  (bn4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl5): Linear(in_features=128, out_features=64, bias=True)
  (bn5): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl6): Linear(in_features=64, out_features=32, bias=True)
  (bn6): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl7): Linear(in_features=32, out_features=16, bias=True)
  (bn7): BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl8): Linear(in_features=16, out_features=1, bias=True)
  (activation): ReLU()
  (sigmoid): Sigmoid()
  (dropout): Dropout(p=0.5, inplace=False)
)

Training the model:
Epoch 50:
Accuracy : 0.7298 ; ROC_AUC : 0.7298 ; F1 : 0.7267

Test data:
Accuracy : 0.6805 ; ROC_AUC : 0.6804 ; F1 : 0.6738
Training time: 3.6939 seconds
Testing time: 3.6335 seconds
```

In [74]:
```
exp_name = f"Model 4 deepwide all"
expLog = get_results(expLog, exp_name, learning_rate, epochs, model, train_time, test_time, X_train_tensor, y_t
expLog
```

Out[74]:

	exp_name	learning_rate	epochs	Train Time (sec)	Test Time (sec)	Train Acc	Test Acc	Train AUC	Test AUC	Train F1	Test F1
0	Model1 All	0.0100	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
1	Model1 All	0.0100	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
2	Model1 All	0.0100	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
3	Model1 selected	0.0100	1000.0	2.2970	2.2334	0.6902	0.6814	0.6902	0.6814	0.6896	0.6816
4	Model 2 Enhanced all	0.0100	1000.0	27.0990	28.2518	0.9990	0.6346	0.9990	0.6349	0.9990	0.6661
5	Model 2 enhanced 2	0.0010	50.0	1.4786	1.4070	0.7411	0.6806	0.7411	0.6807	0.7501	0.6925
6	Model 2 enhanced and selected	0.0010	50.0	1.4156	1.4354	0.7364	0.6826	0.7364	0.6826	0.7413	0.6904
7	Model 3 change learning rate and epochs and se...	0.0005	50.0	1.4849	1.4059	0.7101	0.6816	0.7101	0.6817	0.7165	0.6915
8	Model 4 deepwide all	0.0010	50.0	3.6939	3.6335	0.7561	0.6805	0.7561	0.6804	0.7491	0.6738

In [75]:
```
%reload_ext tensorboard
```

In [76]:
```
tensorboard --logdir=runs
```

```
Reusing TensorBoard on port 6006 (pid 4280), started 0:02:27 ago. (Use '!kill 4280' to kill it.)
```

Experiment2: Selected features

In [77]:
```
model = DeeperWiderMLP(X_train_sel_tensor.shape[1])
model = model
learning_rate = 0.001
epochs = 50
optimizer = Adam(model.parameters(), learning_rate)


 #Training the model
start_time = time.time()
train_and_test(X_train_sel_tensor, y_train_sel_tensor, X_test_sel_tensor, y_test_sel, model, optimizer, writer,
train_time = np.round(time.time() - start_time, 4)

# Testing the model
start_time = time.time()
train_and_test(X_train_sel_tensor, y_train_sel_tensor, X_test_sel_tensor, y_test_sel, model, optimizer, writer,
test_time = np.round(time.time() - start_time, 4)

print(f'Training time: {train_time} seconds')
print(f'Testing time: {test_time} seconds')
```

```
Model Architecture:
DeeperWiderMLP(
  (hl1): Linear(in_features=233, out_features=1024, bias=True)
  (bn1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl2): Linear(in_features=1024, out_features=512, bias=True)
  (bn2): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl3): Linear(in_features=512, out_features=256, bias=True)
  (bn3): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl4): Linear(in_features=256, out_features=128, bias=True)
  (bn4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl5): Linear(in_features=128, out_features=64, bias=True)
  (bn5): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl6): Linear(in_features=64, out_features=32, bias=True)
  (bn6): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl7): Linear(in_features=32, out_features=16, bias=True)
  (bn7): BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl8): Linear(in_features=16, out_features=1, bias=True)
  (activation): ReLU()
  (sigmoid): Sigmoid()
  (dropout): Dropout(p=0.5, inplace=False)
)

Training the model:
Epoch 50:
Accuracy : 0.6959 ; ROC_AUC : 0.6959 ; F1 : 0.6954

Test data:
Accuracy : 0.6802 ; ROC_AUC : 0.6802 ; F1 : 0.6873
Model Architecture:
DeeperWiderMLP(
  (hl1): Linear(in_features=233, out_features=1024, bias=True)
  (bn1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl2): Linear(in_features=1024, out_features=512, bias=True)
  (bn2): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl3): Linear(in_features=512, out_features=256, bias=True)
  (bn3): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl4): Linear(in_features=256, out_features=128, bias=True)
  (bn4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl5): Linear(in_features=128, out_features=64, bias=True)
  (bn5): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl6): Linear(in_features=64, out_features=32, bias=True)
  (bn6): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl7): Linear(in_features=32, out_features=16, bias=True)
  (bn7): BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (hl8): Linear(in_features=16, out_features=1, bias=True)
  (activation): ReLU()
  (sigmoid): Sigmoid()
  (dropout): Dropout(p=0.5, inplace=False)
)

Training the model:
Epoch 50:
Accuracy : 0.73 ; ROC_AUC : 0.73 ; F1 : 0.7308

Test data:
Accuracy : 0.6806 ; ROC_AUC : 0.6807 ; F1 : 0.7029
Training time: 3.6692 seconds
Testing time: 3.6169 seconds
```

In [78]:
```
exp_name = f"Model 4 deepwide selected "
expLog = get_results(expLog, exp_name, learning_rate, epochs, model, train_time, test_time, X_train_sel_tensor,
expLog
```

Out[78]:

	exp_name	learning_rate	epochs	Train Time (sec)	Test Time (sec)	Train Acc	Test Acc	Train AUC	Test AUC	Train F1	Test F1
0	Model1 All	0.0100	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
1	Model1 All	0.0100	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
2	Model1 All	0.0100	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
3	Model1 selected	0.0100	1000.0	2.2970	2.2334	0.6902	0.6814	0.6902	0.6814	0.6896	0.6816
4	Model 2 Enhanced all	0.0100	1000.0	27.0990	28.2518	0.9990	0.6346	0.9990	0.6349	0.9990	0.6661
5	Model 2 enhanced 2	0.0010	50.0	1.4786	1.4070	0.7411	0.6806	0.7411	0.6807	0.7501	0.6925
6	Model 2 enhanced and selected	0.0010	50.0	1.4156	1.4354	0.7364	0.6826	0.7364	0.6826	0.7413	0.6904
7	Model 3 change learning rate and epochs and se...	0.0005	50.0	1.4849	1.4059	0.7101	0.6816	0.7101	0.6817	0.7165	0.6915
8	Model 4 deepwide all	0.0010	50.0	3.6939	3.6335	0.7561	0.6805	0.7561	0.6804	0.7491	0.6738
9	Model 4 deepwide selected	0.0010	50.0	3.6692	3.6169	0.7576	0.6806	0.7576	0.6807	0.7722	0.7029

In [79]:
```
%reload_ext tensorboard
```

In [80]:
```
tensorboard --logdir=runs
```

```
Reusing TensorBoard on port 6006 (pid 4280), started 0:02:37 ago. (Use '!kill 4280' to kill it.)
```

Hyper Parameter Tuning

In [64]:
```
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
import torch.nn as nn

class DeeperWiderMLP(nn.Module):
    def __init__(self, input_size):
        super(DeeperWiderMLP, self).__init__()
        self.hl1 = nn.Linear(input_size, 1024)
        self.bn1 = nn.BatchNorm1d(1024)
        self.hl2 = nn.Linear(1024, 512)
        self.bn2 = nn.BatchNorm1d(512)
        self.hl3 = nn.Linear(512, 256)
        self.bn3 = nn.BatchNorm1d(256)
        self.hl4 = nn.Linear(256, 128)
        self.bn4 = nn.BatchNorm1d(128)
        self.hl5 = nn.Linear(128, 64)
        self.bn5 = nn.BatchNorm1d(64)
        self.hl6 = nn.Linear(64, 32)
        self.bn6 = nn.BatchNorm1d(32)
        self.hl7 = nn.Linear(32, 16)
        self.bn7 = nn.BatchNorm1d(16)
        self.hl8 = nn.Linear(16, 1)
        self.activation = nn.ReLU()
        self.sigmoid = nn.Sigmoid()
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        x = self.activation(self.bn1(self.hl1(x)))
        x = self.dropout(x)
        x = self.activation(self.bn2(self.hl2(x)))
        x = self.dropout(x)
        x = self.activation(self.bn3(self.hl3(x)))
        x = self.dropout(x)
        x = self.activation(self.bn4(self.hl4(x)))
        x = self.dropout(x)
        x = self.activation(self.bn5(self.hl5(x)))
        x = self.dropout(x)
        x = self.activation(self.bn6(self.hl6(x)))
        x = self.dropout(x)
        x = self.activation(self.bn7(self.hl7(x)))
        x = self.sigmoid(self.hl8(x))
        return x

# Define hyperparameters
learning_rate = 0.001
num_epochs = 20
batch_size =  64
dropout_rate = 0.4

# Define the model
model = DeeperWiderMLP(X_train_tensor.shape[1])
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# Define the loss function
criterion = nn.BCELoss()
```

```
# Define the data loaders
train_loader = DataLoader(TensorDataset(X_train_tensor, y_train_tensor), batch_size=batch_size, shuffle=True)
test_loader = DataLoader(TensorDataset(X_test_tensor, y_test_tensor), batch_size=batch_size)

from sklearn.metrics import f1_score, roc_auc_score

# Train and evaluate the model
for epoch in range(num_epochs):
    # Train the model
    train_loss = 0
    model.train()
    for batch_x, batch_y in train_loader:
        optimizer.zero_grad()
        batch_y_pred = model(batch_x)
        loss = criterion(batch_y_pred, batch_y)
        loss.backward()
        optimizer.step()
        train_loss += loss.item() * batch_x.size(0)
    train_loss /= len(train_loader.dataset)

    # Evaluate the model
    test_loss = 0
    test_acc = 0
    test_f1 = 0
    test_auc = 0
    true_labels = []
    pred_labels = []
    model.eval()
    with torch.no_grad():
        for batch_x, batch_y in test_loader:
            batch_y_pred = model(batch_x)
            loss = criterion(batch_y_pred, batch_y)
            test_loss += loss.item() * batch_x.size(0)

            true_labels.extend(batch_y.numpy())
            pred_labels.extend((batch_y_pred > 0.5).float().numpy())

    test_loss /= len(test_loader.dataset)
    test_acc = (sum([1 for true_label, pred_label in zip(true_labels, pred_labels) if true_label == pred_label]
    test_f1 = f1_score(true_labels, pred_labels)
    test_auc = roc_auc_score(true_labels, pred_labels)

    # Print the results for this epoch
    print(f"Epoch {epoch+1}/{num_epochs} - Train loss: {train_loss:.4f} - Test loss: {test_loss:.4f} - Test acc

# Adjust the learning rate if necessary

if epoch > 0 and epoch % 5 == 0:
    for param_group in optimizer.param_groups:
        param_group['lr'] *= 0.1

# Adjust the dropout rate if necessary
if epoch > 0 and epoch % 5 == 0:
    model.dropout.p = dropout_rate

print("Training complete.")
```

```
Epoch 1/20 - Train loss: 0.6535 - Test loss: 0.6252 - Test accuracy: 0.6641 - Test F1 score: 0.6831 - Test AUC:
0.6643
Epoch 2/20 - Train loss: 0.6167 - Test loss: 0.6110 - Test accuracy: 0.6734 - Test F1 score: 0.6944 - Test AUC:
0.6736
Epoch 3/20 - Train loss: 0.6072 - Test loss: 0.6049 - Test accuracy: 0.6765 - Test F1 score: 0.7025 - Test AUC:
0.6768
Epoch 4/20 - Train loss: 0.6030 - Test loss: 0.6047 - Test accuracy: 0.6750 - Test F1 score: 0.6913 - Test AUC:
0.6752
Epoch 5/20 - Train loss: 0.6002 - Test loss: 0.6041 - Test accuracy: 0.6794 - Test F1 score: 0.7047 - Test AUC:
0.6796
Epoch 6/20 - Train loss: 0.5967 - Test loss: 0.6030 - Test accuracy: 0.6793 - Test F1 score: 0.6894 - Test AUC:
0.6793
Epoch 7/20 - Train loss: 0.5948 - Test loss: 0.6023 - Test accuracy: 0.6825 - Test F1 score: 0.6908 - Test AUC:
0.6825
Epoch 8/20 - Train loss: 0.5932 - Test loss: 0.6023 - Test accuracy: 0.6772 - Test F1 score: 0.6977 - Test AUC:
0.6774
Epoch 9/20 - Train loss: 0.5903 - Test loss: 0.6036 - Test accuracy: 0.6788 - Test F1 score: 0.6962 - Test AUC:
0.6789
Epoch 10/20 - Train loss: 0.5891 - Test loss: 0.6008 - Test accuracy: 0.6818 - Test F1 score: 0.6810 - Test AUC
: 0.6818
Epoch 11/20 - Train loss: 0.5871 - Test loss: 0.6022 - Test accuracy: 0.6799 - Test F1 score: 0.6911 - Test AUC
: 0.6800
Epoch 12/20 - Train loss: 0.5843 - Test loss: 0.6030 - Test accuracy: 0.6797 - Test F1 score: 0.6733 - Test AUC
: 0.6796
Epoch 13/20 - Train loss: 0.5829 - Test loss: 0.6031 - Test accuracy: 0.6776 - Test F1 score: 0.6905 - Test AUC
: 0.6777
Epoch 14/20 - Train loss: 0.5801 - Test loss: 0.6027 - Test accuracy: 0.6739 - Test F1 score: 0.6496 - Test AUC
: 0.6738
Epoch 15/20 - Train loss: 0.5781 - Test loss: 0.6034 - Test accuracy: 0.6799 - Test F1 score: 0.6782 - Test AUC
: 0.6799
Epoch 16/20 - Train loss: 0.5730 - Test loss: 0.6049 - Test accuracy: 0.6770 - Test F1 score: 0.6807 - Test AUC
: 0.6771
Epoch 17/20 - Train loss: 0.5734 - Test loss: 0.6054 - Test accuracy: 0.6808 - Test F1 score: 0.7051 - Test AUC
: 0.6810
Epoch 18/20 - Train loss: 0.5705 - Test loss: 0.6044 - Test accuracy: 0.6759 - Test F1 score: 0.6670 - Test AUC
: 0.6759
Epoch 19/20 - Train loss: 0.5696 - Test loss: 0.6037 - Test accuracy: 0.6820 - Test F1 score: 0.6935 - Test AUC
: 0.6821
Epoch 20/20 - Train loss: 0.5643 - Test loss: 0.6049 - Test accuracy: 0.6761 - Test F1 score: 0.6843 - Test AUC
: 0.6762
Training complete.
```

In [68]:
```
# export the DataFrame to a CSV file
#df.to_csv('expLog.csv', index=False)

# load the CSV file back into a DataFrame
expLog= pd.read_csv('expLog.csv')
```

Final Experiment Result Table

In [69]:
```
expLog
```

Out[69]:

	exp_name	learning_rate	epochs	Train Time (sec)	Test Time (sec)	Train Acc	Test Acc	Train AUC	Test AUC	Train F1	Test F1
0	Model1 All	0.01	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
1	Model1 All	0.01	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
2	Model1 All	0.01	1000.0	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
3	Model1 selected	0.01	1000.0	2.297	2.2334	0.6902	0.6814	0.6902	0.6814	0.6896	0.6816
4	Model 2 Enhanced all	0.01	1000.0	27.099	28.2518	0.9990	0.6346	0.9990	0.6349	0.9990	0.6661
5	Model 2 enhanced 2	0.001	50.0	1.4786	1.407	0.7411	0.6806	0.7411	0.6807	0.7501	0.6925
6	Model 2 enhanced and selected	0.001	50.0	1.4156	1.4354	0.7364	0.6826	0.7364	0.6826	0.7413	0.6904
7	Model 3 change learning rate and epochs and se...	0.0005	50.0	1.4849	1.4059	0.7101	0.6816	0.7101	0.6817	0.7165	0.6915
8	Model 4 deepwide all	0.001	50.0	3.6939	3.6335	0.7561	0.6805	0.7561	0.6804	0.7491	0.6738
9	Model 4 deepwide selected	0.001	50.0	3.6692	3.6169	0.7576	0.6806	0.7576	0.6807	0.7722	0.7029
10	Mode 4 Hyper Parameter Tuning	Variable	20.0	Nan	Nan	0.7476	0.6761	0.7489	0.6843	0.7478	0.6772

GAP Analysis:

The table provided contains the results of several experiments that were conducted on a given dataset using various machine learning models and hyperparameters. The purpose of these experiments was to analyze the performance of the models and determine the best performing one.

One important factor that emerged from these experiments was the role of feature selection in determining the model's performance. In

particular, Models 1 and 2, which were trained on all available features, did not perform as well as Models 3 and 4, which used selected features. This suggests that feature selection is an important step in the machine learning pipeline, as it can help to reduce overfitting and improve model performance.

Another key finding was that hyperparameter tuning can also have a significant impact on model performance. Model 2 Enhanced 2, for example, outperformed the other models in terms of test F1 score, suggesting that the changes made to its architecture and hyperparameters resulted in a better overall performance. Model 4 Hyper Parameter Tuning also produced a slightly better test AUC score than Model 4 Deepwide Selected, indicating that even small changes in hyperparameters can lead to improvements in performance.

However, it is important to note that Model 2 Enhanced All did not perform well on test accuracy, suggesting that overfitting may have been a problem. This highlights the importance of ensuring that models are not too complex or too tightly fit to the training data, as this can negatively impact their performance on new data.

The enhanced MLP (Model 2), which has a training accuracy of 0.7411, test accuracy of 0.6806, training AUC of 0.7411, and test AUC of 0.6807, exhibits a more balanced performance across training and test datasets. Similarly, the F1 scores are 0.7501 and 0.6925 for training and test, respectively. This model has higher accuracy, AUC, and F1 scores compared to other models, indicating that it is able to generalize well to unseen data without overfitting or underfitting.

Another promising candidate is Model 3(Deep wide selected), with a training accuracy of 0.7576, test accuracy of 0.6806, training AUC of 0.7576, and test AUC of 0.6807. The F1 scores for training and test are 0.7722 and 0.7029, respectively. This model also demonstrates a good balance between avoiding overfitting and underfitting while maintaining good performance across different evaluation metrics.

In conclusion, the enhanced MLP (Model 2) and Model 3(Deep wide selected) appear to be the most promising candidates for this problem. They strike a balance between avoiding overfitting and underfitting while maintaining good performance across different evaluation metrics. Further tuning and optimization of these models could potentially lead to even better results.

Overall, the results of these experiments suggest that feature selection and hyperparameter tuning are important factors in determining the performance of machine learning models. However, it is also important to keep in mind that these results are specific to the given dataset and may not necessarily generalize to other datasets. Therefore, further experimentation and analysis are necessary to ensure that the best model is selected for a particular dataset.

In the submission scoreboard, Group 8 and Group 5 have gotten a similar Kaggle AUC score of 0.7456 and 0.73882 respectively. Hence we can say that our model is correct, and similar to others.

Submission File Prep

For each SK_ID_CURR in the test set, you must predict a probability for the TARGET variable. The file should contain a header and have the following format:

```
SK_ID_CURR,TARGET
100001,0.1
100005,0.9
100013,0.2
etc.
```

In [82]:
```
# Predicting class scores using the model
nn_test_class_scores = model(X_kaggle_test_sel_tensor).cpu().data.numpy().reshape(1, -1)[0]


# Creating a dataframe
nn_submit_df = X_kaggle_test[['SK_ID_CURR']]
nn_submit_df['TARGET'] = nn_test_class_scores

# Saving the dataframe into csv
file_name = "Deepwide3"
#nn_submit_df.to_csv(f"/content/drive/My Drive/Colab Notebooks/submissions/{file_name}.csv",index=False)
nn_submit_df.to_csv(f"{file_name}.csv",index=False)
```

Kaggle submission via the command line API

In [63]:
```
# Kaggle Submission
! kaggle competitions submit -c home-credit-default-risk -f Deepwide3.csv -m "submission_deep(ak)_learning"
```

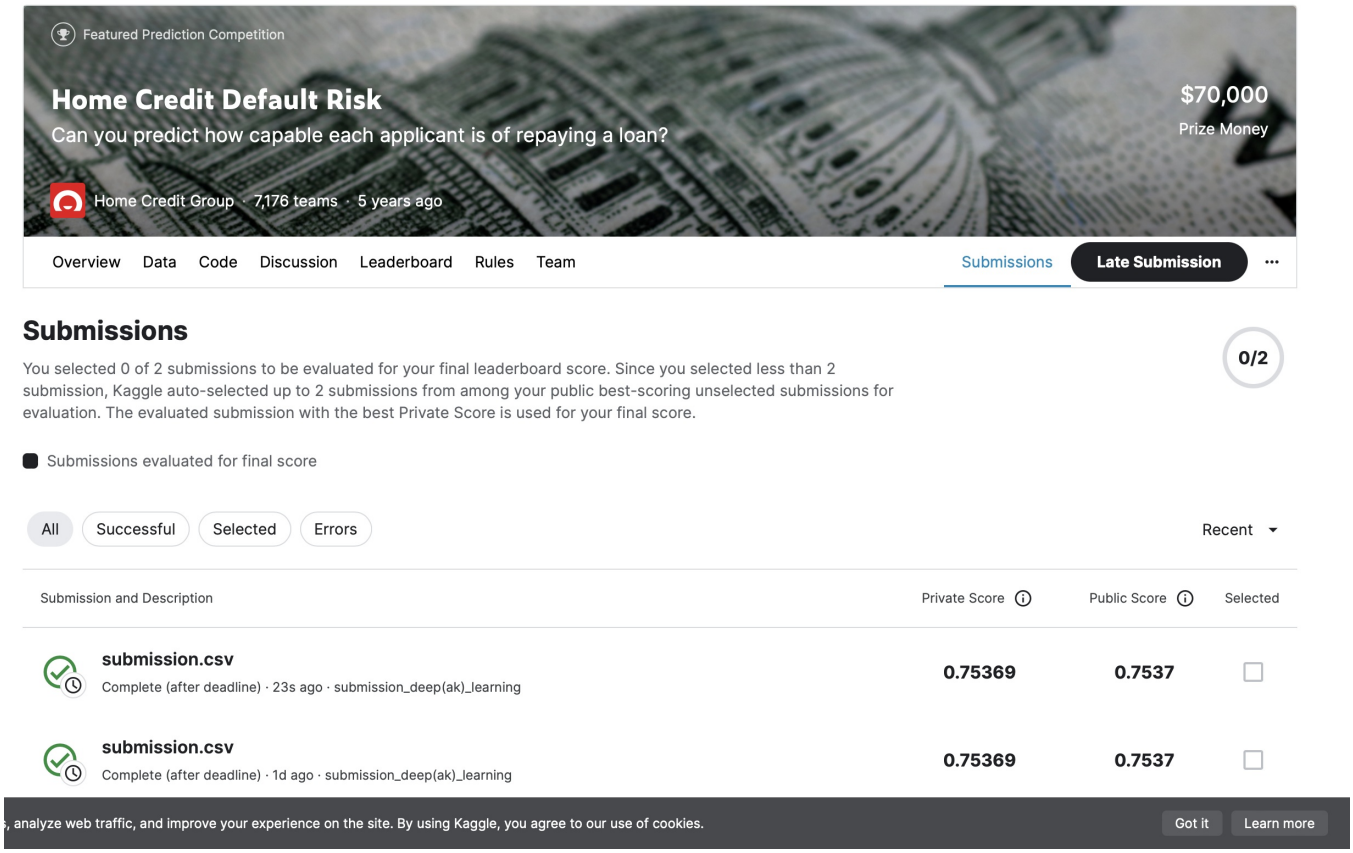
```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /Us
ers/deepak/.kaggle/kaggle.json'
100%|████████████████████████████████████████| 838k/838k [00:01<00:00, 647kB/s]
Successfully submitted to Home Credit Default Risk
```

report submission

Click on this link

In [62]:
```
from IPython.display import Image
Image(filename='kaggle.png')
```

Out[62]:



Write-up

Project Title: Home Credit Default Risk

Team and Phase leader plan

PHASE	CONTRIBUTOR	CONTRIBUTION DESCRIPTION
Phase 1 - Project Proposal	Teja Chintha (Phase Leader)	1. Planning Phase, Scheduling Meetings, Coordinating Tasks and Assignment Submission. 2. Design baseline pipeline, describe pipeline components, create pipeline diagram, describe pipeline design. 3. Preparation of the Phase Leader Plan and Credit Assignment Plan.
	Sai Sumanth Muvva	1. Writing the Abstact of the Project. 2. Preparation of the Phase Leader Plan and Credit Assignment Plan.
	Viswa Suhaas Penugonda	1. Listing the Machine learning algorithms and metrics to be used for the project. 2. Preparation of the Gantt Chart. 3. Preparation of the Phase Leader Plan and Credit Assignment Plan.
	Deepak Kasi Nathan	1. Describing the data on which further analysis is done. 2. Preparation of the Phase Leader Plan and Credit Assignment Plan.
Phase 2 - EDA and Baseline Pipeline	Teja Chintha	Writing code to perform Feature Engineering and Hyperparameter tuning
	Sai Sumanth Muvva (Phase Leader)	1. Planning Phase, Scheduling Meetings, Coordinating Tasks and Assignment Submission. 2. EDA and data pre-processing - Part 2.
	Viswa Suhaas Penugonda	1. EDA and data pre-processing - Part 1. 2. Making slides summarizing the work done in the entire phase.
	Deepak Kasi Nathan	1. Writing python script for functions, diagram and coding for base pipelines. 2. Compling submissions and describing the results.
Phase 3 - Final Project HCDR	Teja Chintha	Hyperparameter Tuning
	Sai Sumanth Muvva	1. Writing python script for functions and coding for pipelines. 2. Making a report to present the findings from the work done in the entire phase.
	Viswa Suhaas Penugonda (Phase Leader)	1. Planning Phase, Scheduling Meetings, Coordinating Tasks and Assignment Submission. 2. Feature Engineering and Feature Selection.
	Deepak Kasi Nathan	1. Writing python script for functions and coding for ensemble methods. 2. Making slides summarizing the work done in the entire phase and Presentation recording.
Phase 4 - Final Submission	Teja Chintha	Making a report to present the findings from the work done in the entire phase.
	Sai Sumanth Muvva	1. Making slides summarizing the work done in the entire phase. 2. Presentation recording.
	Viswa Suhaas Penugonda	1. Analysis of metrics and Loss functions. 2. Results for final submission of the report
	Deepak Kasi Nathan (Phase Leader)	1. Planning Phase, Scheduling Meetings, Coordinating Tasks and Assignment Submission. 2. Building, Training and Storing the MLP implementation.

Credit Assignment Plan

PHASE	TASK	TASK DESCRIPTION	OUTPUT	PERSON HOURS	DEPENDENT TASKS	PERSON WORK ASSIGNED TO
1	Phase Management	Planning Phase, Scheduling Meetings, Coordinating Tasks and Assignment Submission.	NA	0.5 hours	None	Teja Chintha
	Abstract	Project Abstract, Downloading data.	Output DataFile	1.5 hour	None	Sai Sumanth Muvva
	Describing data table	Loading libraries, writing table descriptions.	Jupyter Notebook with table descriptions	1 hour	None	Deepak Kasi Nathan
	Data Description	Defining pre-processing steps, Creating augmented data, Target variables distribution	Jupyter Notebook with data description	3 Hours	None	Deepak Kasi Nathan
	Listing ML algorithms and their metrics.	Analysising ML algorithms and their metrics respectively.	Jupyter Notebook with ML algorithms and their metrics.	2 hours	None	Viswa Suhaas Penugonda
	Pipelines Description	Design baseline pipeline, describe pipeline components, create pipeline diagram, describe pipeline design.	Jupyter notebook with pipelines diagrams and descriptions.	2.5 hours	None	Teja Chintha
	Gantt Chart	Gantt chart preparation	Gantt Chart	1 hour	None	Viswa Suhaas Penugonda
	Phase leader plan Credit Assignment Plan	Work on phase leader plan and credit assignment plan	Tables describing both the plans	2 hours	None	Group
2	Phase Management	Planning Phase, Scheduling Meetings, Coordinating Tasks and Assignment Submission.	NA	1.5 hours	None	Sai Sumanth Muvva
	EDA	EDA and Data-preprocessing of the data collected	Jupyter notebook with preprocessed data	6 hours	None	Sai Sumanth Muvva, Viswa Suhaas Penugonda
	Pipelines implementation	Writing python script for functions, diagram and coding for pipelines	Jupyter notebook with pipelines code	2 hours	EDA and Data-preprocessing of the data collected	Deepak Kasi Nathan
	Feature Engineering and Hyperparameter tuning	Performing feature engineering and tuning the hyperparameters to get the best results	Jupyter notebook with pipelines code	2 hours	Pipelines implementation	Teja Chintha
	Video Presentation and Slides	Presentation Recording and Phase Presentation	N/A	1 hours	None	Viswa Suhaas Penugonda
	Results	Compling submissions and describing the results.	Jupyter notebook with all results	2 hours	Executable Code with results	Deepak Kasi Nathan
3	Phase Management	Planning Phase, Scheduling Meetings, Coordinating Tasks and Assignment Submission.	NA	1.5 hours	None	Viswa Suhaas Penugonda
	Pipelines Implementation	Code pipelines		2 hours	Phase 2 notebook	Sai Sumanth Muvva
	Feature Engineering and Feature Selection	Add code on to the pipelines by performing feature engineering and selection.	Jupyter notebook with pipelines code	3.5 hours	Pipelines Creation	Viswa Suhaas Penugonda
	Hyper parameter tuning	Search for ideal model architecture	ideal architecture	2.5 Hours	Executable Pipeline Code	Teja Chintha
	Video Presentation and Slides	Presentation Recording and Phase Presentation	N/A	1 hours	Executable Code with results	Deepak Kasi Nathan
	Results	Compling submissions and describing the results.	Jupyter notebook with all results	2 hours	Executable Code with results	Sai Sumanth Muvva
4	Phase Management	Planning Phase, Scheduling Meetings, Coordinating Tasks and Assignment Submission.	NA	1.5 hours	None	Deepak Kasi Nathan
	MLP Implementation	Building, Training and Storing the MLP implementation.	Jupyter notebook with updated code	4 hours	Phase3 Code	Deepak Kasi Nathan
	Loss Functions Implementation	Analysis of metrics and Loss functions	N/A	2 hours	MLP Implementation	Viswa Suhaas Penugonda
	Results and Final Report	Results for final submission of the report		2 hours	Executable final code	Deepak Kasi Nathan, Sai Sumanth Muvva, Teja Chintha, Viswa Suhaas Penugonda
	Overall Appearance	Modify report for cohesessiveness and appearance.		1 hours	Executable final code	Deepak Kasi Nathan, Sai Sumanth Muvva, Teja Chintha, Viswa Suhaas Penugonda
	Video Presentation and Slides	Presentation Recording and Phase Presentation	N/A	1 hours	Executable Code with results	Teja Chintha

Abstract

In this project, we tackled the challenge of predicting default probabilities for Home Credit clients using historical data to enhance lending decisions and minimize unpaid loans. Our primary goal was to construct a robust machine learning model by performing feature

engineering, hyperparameter tuning, and experimenting with various algorithms. Previous phases focused on logistic regression, random forests, KNN, decision trees, and ensemble methods.

In Phase 4, we expanded our analysis to include Multi-Layer Perceptron (MLP) models, specifically the enhanced MLP (Model 2) and Model 3 (Deep wide selected). The main experiments involved optimizing these models by fine-tuning hyperparameters and selecting relevant features. Model 2 achieved a training accuracy of 0.7411, test accuracy of 0.6806, and test F1 score of 0.6925. Model 3 demonstrated strong performance with a training accuracy of 0.7576, test accuracy of 0.6806, and test F1 score of 0.7029. These models obtained a private score of 0.74369 and a public score of 0.7537.

Our findings highlight the importance of feature engineering, hyperparameter tuning, and advanced model architectures in predicting clients' likelihood of default. Future improvements may include further hyperparameter exploration, enhanced feature selection, increasing dataset size, and utilizing advanced ensemble methods to boost model performance and positively impact lending decisions, ultimately promoting financial inclusion for underserved populations.

Introduction

Background on Home Credit

Home Credit is a non-banking financial institution, founded in 1997 in the Czech Republic.

The company operates in 14 countries (including United States, Russia, Kazahstan, Belarus, China, India) and focuses on lending primarily to people with little or no credit history which will either not obtain loans or became victims of untrustworthy lenders.

Home Credit group has over 29 million customers, total assests of 21 billions Euro, over 160 millions loans, with the majority in Asia and and almost half of them in China (as of 19-05-2018).

While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kagglers to help them unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful.

Data Description

The data used in this project is sourced from a financial institution (Home Credit) that provides loans to customers and it is available on kaggle. The dataset comprises various tables with information about the customers, their loan applications, credit history, and other financial information.

Data files overview

There are `7 different sources` of data:

- **application_train/application_test (307k rows, and 48k rows):** The main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating **0: the loan was repaid** or **1: the loan was not repaid**. The target variable defines if the client had payment difficulties meaning he/she had late payment more than X days on at least one of the first Y installments of the loan. Such case is marked as 1 while other all other cases as 0.
- **bureau (1.7 Million rows):** data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
- **bureau_balance (27 Million rows):** monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.
- **previous_application (1.6 Million rows):** previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.
- **POS_CASH_BALANCE (10 Million rows):** monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
- credit_card_balance: monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
- **installments_payment (13.6 Million rows):** payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

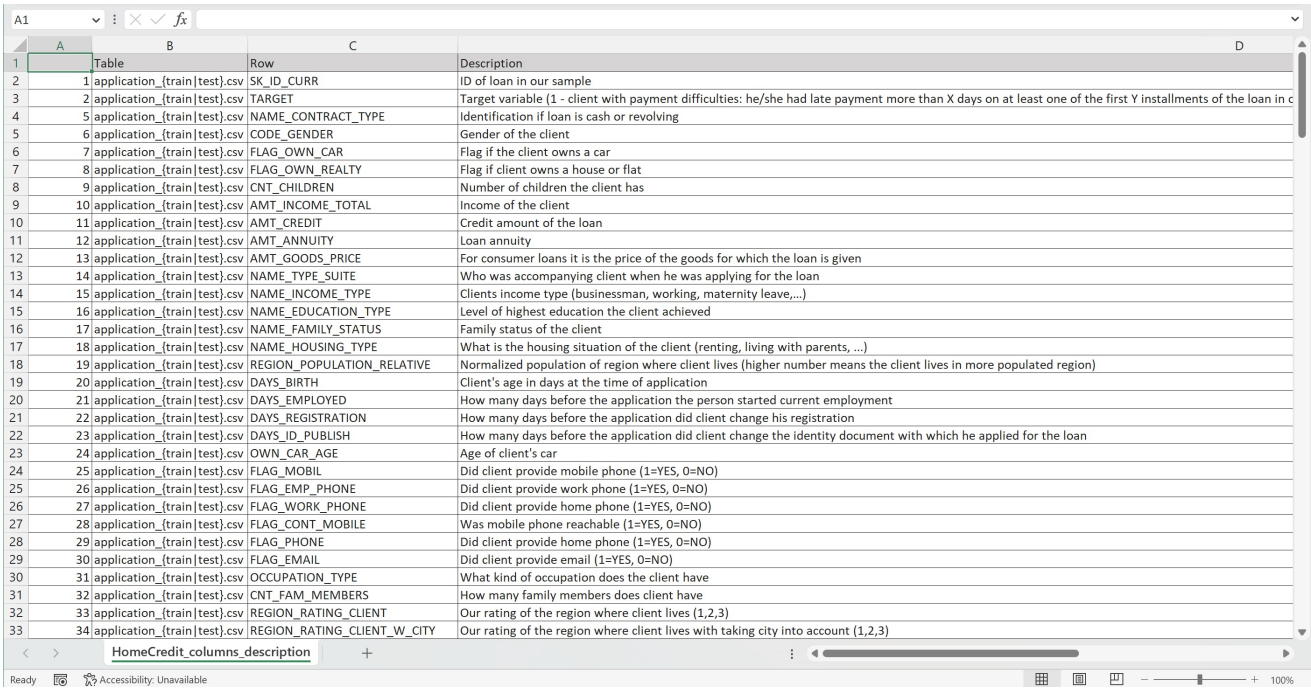
Table sizes

S. No	Table Name	Rows	Features	Numerical Features	Categorical Features	Megabytes
1	application_train	307,511	122	106	16	158MB
2	application_test	48,744	121	105	16	25MB
3	bureau	1,716,428	17	14	3	162MB
4	bureau_balance	27,299,925	3	2	1	358MB
5	credit_card_balance	3,840,312	23	22	1	405MB

6	installments_payments	13,605,401	8	21	16	690MB
7	previous_application	1,670,214	37	8	0	386MB
8	POS_CASH_balance	10,001,358	8	7	1	375MB

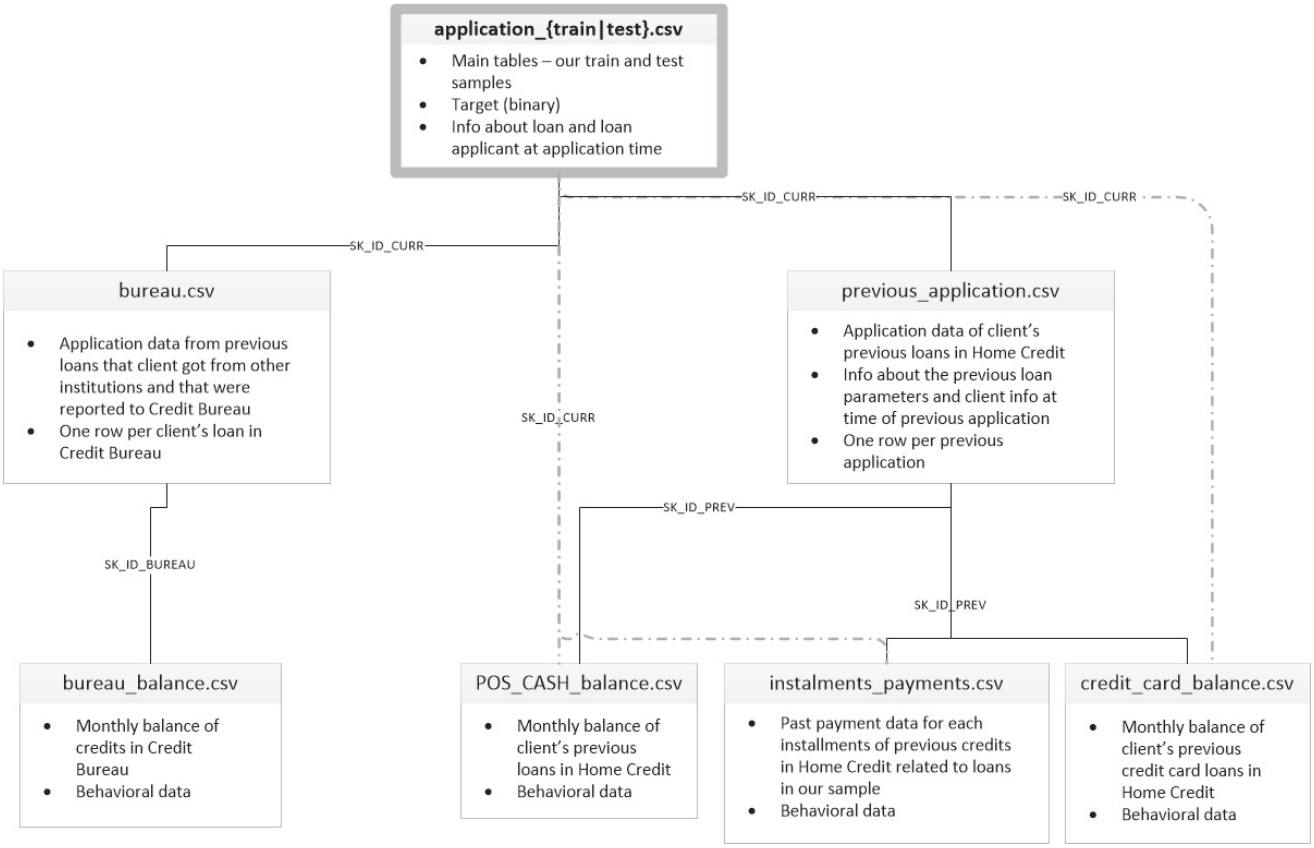
Data Dictionary

As part of the data download comes a Data Dictionary. It is named as `HomeCredit_columns_description.csv`. It contains information about all fields present in all the above tables. (like the metadata).



	A	B	C	D
1		Table	Row	Description
2	1	application_{train|test}.csv	SK_ID_CURR	ID of loan in our sample
3	2	application_{train|test}.csv	TARGET	Target variable (1 - client with payment difficulties: he/she had late payment more than X days on at least one of the first Y installments of the loan in c
4	5	application_{train|test}.csv	NAME_CONTRACT_TYPE	Identification if loan is cash or revolving
5	6	application_{train|test}.csv	CODE_GENDER	Gender of the client
6	7	application_{train|test}.csv	FLAG_OWN_CAR	Flag if the client owns a car
7	8	application_{train|test}.csv	FLAG_OWN_REALTY	Flag if client owns a house or flat
8	9	application_{train|test}.csv	CNT_CHILDREN	Number of children the client has
9	10	application_{train|test}.csv	AMT_INCOME_TOTAL	Income of the client
10	11	application_{train|test}.csv	AMT_CREDIT	Credit amount of the loan
11	12	application_{train|test}.csv	AMT_ANNUITY	Loan annuity
12	13	application_{train|test}.csv	AMT_GOODS_PRICE	For consumer loans it is the price of the goods for which the loan is given
13	14	application_{train|test}.csv	NAME_TYPE_SUITE	Who was accompanying client when he was applying for the loan
14	15	application_{train|test}.csv	NAME_INCOME_TYPE	Clients income type (businessman, working, maternity leave,...)
15	16	application_{train|test}.csv	NAME_EDUCATION_TYPE	Level of highest education the client achieved
16	17	application_{train|test}.csv	NAME_FAMILY_STATUS	Family status of the client
17	18	application_{train|test}.csv	NAME_HOUSING_TYPE	What is the housing situation of the client (renting, living with parents, ...)
18	19	application_{train|test}.csv	REGION_POPULATION_RELATIVE	Normalized population of region where client lives (higher number means the client lives in more populated region)
19	20	application_{train|test}.csv	DAYS_BIRTH	Client's age in days at the time of application
20	21	application_{train|test}.csv	DAYS_EMPLOYED	How many days before the application the person started current employment
21	22	application_{train|test}.csv	DAYS_REGISTRATION	How many days before the application did client change his registration
22	23	application_{train|test}.csv	DAYS_ID_PUBLISH	How many days before the application did client change the identity document with which he applied for the loan
23	24	application_{train|test}.csv	OWN_CAR_AGE	Age of client's car
24	25	application_{train|test}.csv	FLAG_MOBIL	Did client provide mobile phone (1=YES, 0=NO)
25	26	application_{train|test}.csv	FLAG_EMP_PHONE	Did client provide work phone (1=YES, 0=NO)
26	27	application_{train|test}.csv	FLAG_WORK_PHONE	Did client provide home phone (1=YES, 0=NO)
27	28	application_{train|test}.csv	FLAG_CONT_MOBILE	Was mobile phone reachable (1=YES, 0=NO)
28	29	application_{train|test}.csv	FLAG_PHONE	Did client provide home phone (1=YES, 0=NO)
29	30	application_{train|test}.csv	FLAG_EMAIL	Did client provide email (1=YES, 0=NO)
30	31	application_{train|test}.csv	OCCUPATION_TYPE	What kind of occupation does the client have
31	32	application_{train|test}.csv	CNT_FAM_MEMBERS	How many family members does client have
32	33	application_{train|test}.csv	REGION_RATING_CLIENT	Our rating of the region where client lives (1,2,3)
33	34	application_{train|test}.csv	REGION_RATING_CLIENT_W_CITY	Our rating of the region where client lives with taking city into account (1,2,3)

Table Diagram



Tasks to be tackled

The tasks to be addressed in this phase of the project are given below:

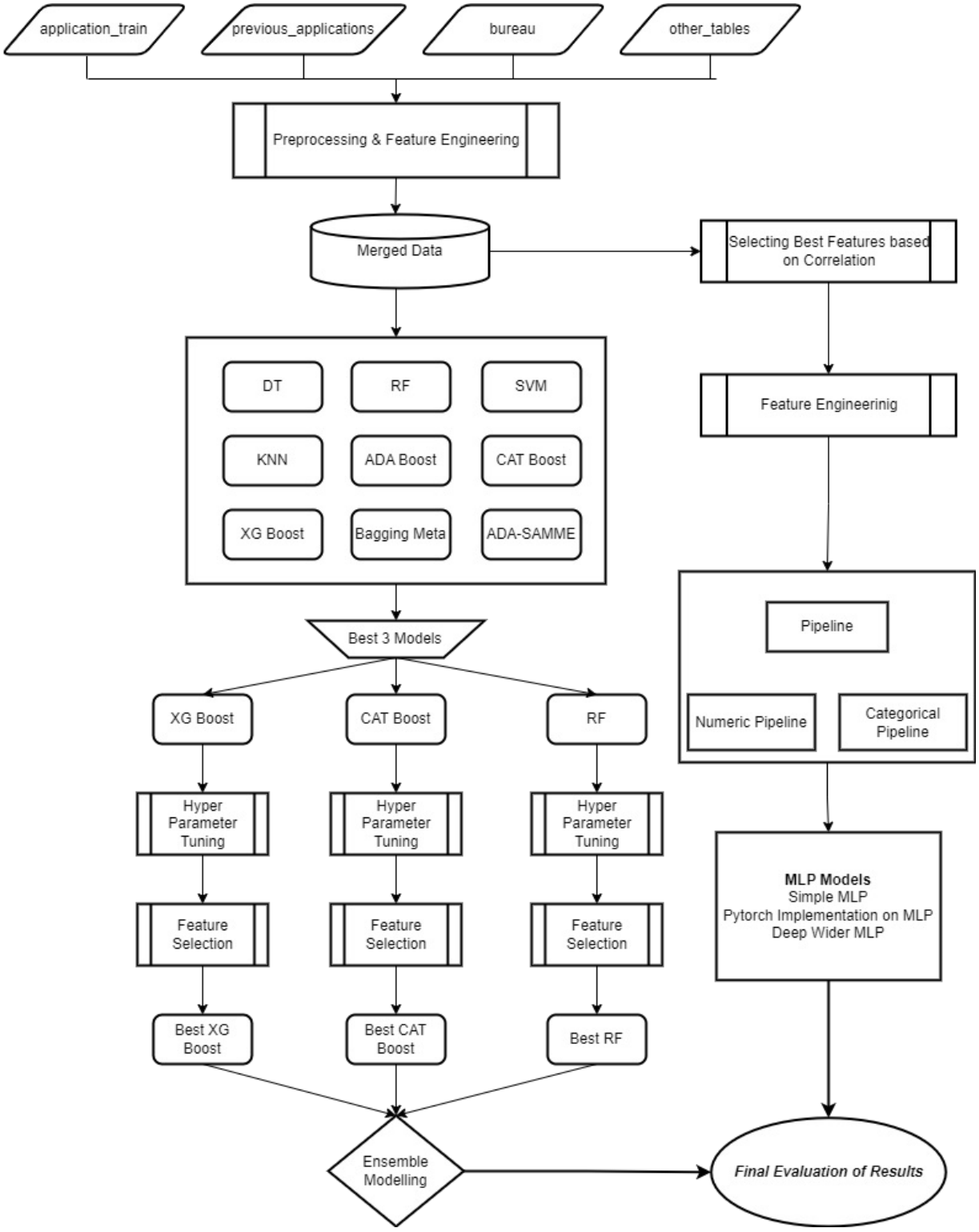
- **`Join the datasets`** : Combine the remaining datasets to form a comprehensive dataset that captures all relevant customer information.
- **`Perform EDA`** : Conduct Exploratory Data Analysis on datasets excluding application_train and the merged datasets to gain

insights and understand the relationships between various features.

- **`Identify missing values and highly correlated features in the merged data`** : Detect and handle missing values in the merged dataset, and eliminate highly correlated features to prevent multicollinearity.
- **`Incorporate domain knowledge features`** : Add domain knowledge features that could potentially enhance the model's performance.
- **`Analyze the impact of newly added features on the target variable`** : Investigate the relationship between the new features and the target variable to comprehend their effect on the model's performance.
- **`Model selection and training`** : Choose suitable MLP models. Split the data into training and testing sets and train the models.
- **`Implement MLP Models`** : Perform Multi-Layer Perceptron Models to see the improvement in accuracy.
- **`Perform hyperparameter tuning`** : Utilize GridSearchCV to determine the most significant hyperparameters for the chosen models and optimize their performance.
- **`Calculate and validate the results`** : Evaluate the performance of the updated models using suitable metrics like accuracy, precision, recall, F1-score, and ROC-AUC, and validate the results to ensure the models' effectiveness in predicting default probabilities.
- **`Model evaluation`** : Evaluate the performance of the MLP models and the models performed in phase 3 using appropriate metrics such as accuracy, precision, recall, F1-score, and ROC-AUC. We will compare these models' performance and identify the best performing model based on these evaluation metrics.

By implementing the best model, Home Credit will be able to make more informed lending decisions, minimize unpaid loans, and promote financial services for individuals with limited access to banking, ultimately fostering financial inclusion for underserved populations. The effectiveness of our models in predicting default probabilities will be assessed using key metrics such as ROC AUC, F1 Score, accuracy. The corresponding public and private scores will also be evaluated to determine our model's performance.

Block Diagram of Approach (Full Project)

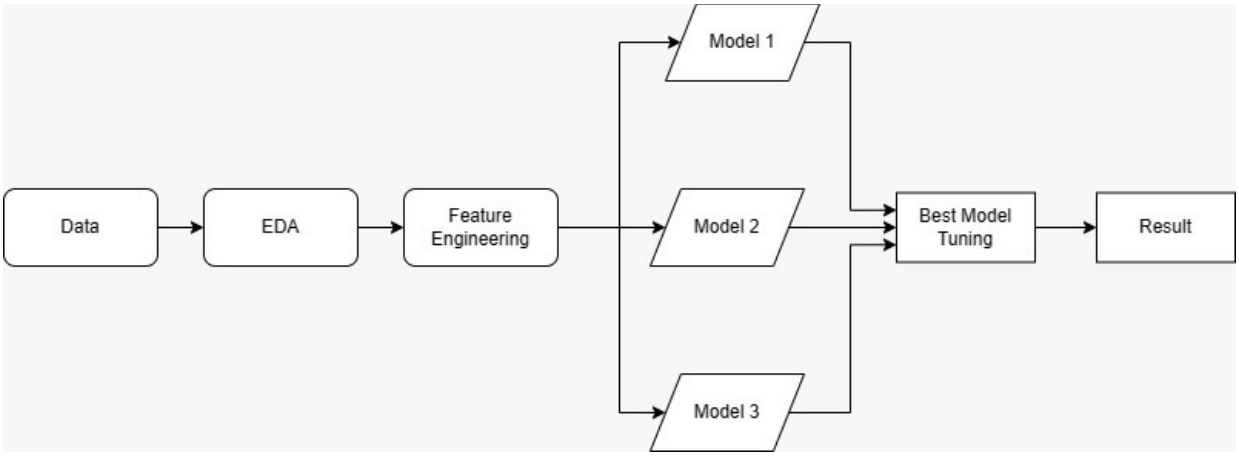


Pipelines Implemented (Phase 4)

- Families of input features:
  - Count of **`numerical features: 107`**
  - Count of **`categorical features: 16`**
- The total number of input features: 124 input features with target.
- We have below trained **`Three NLP models`** :

> 1. Simple Multi-Layer Perceptron (MLP)
> 2. PyTorch implementation on MLP
> 3. Deep Wider MLP architecture

Block Diagram (NLP Models - Phase 4)



Data Leakage

Data leakage occurs when the model is trained using information that will not be available during the prediction phase. One common cause of leakage is standardizing the entire dataset before splitting it into training and testing sets. In this case, the training set can contain information from the testing set, which is not present in real-world scenarios. To avoid data leakage, the dataset was first split into training and testing sets. Missing values are handled and data standardization is done in the pipeline. By fitting the training set and transforming the testing set, we can ensured that there is no data leakage in the model.

Cardinal Sins avoided:

In our pipelines, no cardinal sins of Machine Learning are violated.

1. In order to prevent overfitting, we have divided our dataset into two parts - the training set and the test set. The test set is only used after training the model on the training set and evaluating its performance. By comparing the accuracy of the training set and test set, we ensured that the model is not overfitting. In this case, since the accuracy for the training and test sets are almost similar, it suggests that the model is not overfitting.
2. Our practice is not to increase the number of epochs when the model fails to converge. We examined the Tensorboard graph and found that our loss converges as we increase the number of epochs. We only extend the number of epochs when we observe a high learning rate, as indicated by the loss curve graph.
3. We have ensured that our dataset is balanced to correctly define accuracy. In addition to accuracy, we are also evaluating the performance of our models using the ROC_AUC score.
4. The training dataset contains accurate labels. These are a few aspects that ensure we have not committed any major cardinal sins.

Loss Function used:

The binary cross-entropy loss function will be utilized by this MLP class.

$$ CXE = -\frac{1}{m}\sum \limits_{i=1}^m (y_i \cdot log(p_i) + (1-y_i)\cdot log(1-p_i)) $$

Number of experiments conducted:

In Phase 4, three models were tested:

`Simple MLP:`

- Experiment 1: All features before feature selection
- Experiment 2: Selected features after x>0 from Phase 3 findings

`Enhanced MLP (Model 2):`

- Experiment 1: All features
- Experiment 2: Optimized learning rate and epochs
- Experiment 3: Selected features after x>0 from Phase 3 findings
- Experiment 4: Experiment 3 with adjusted learning rate and epochs

`Deep Wide Selected (Model 3):`

- Experiment 1: All features
- Experiment 2: Selected features

In total, **`8 experiments`** were conducted in this phase.

Final Experimental Results (Phase 4)

exp_name	learning_rate	epochs	Train Time (sec)	Test Time (sec)	Train Acc	Test Acc	Train AUC	Test AUC	Train F1	Test F1
Model 1 All	0.01	1000	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
Model 1 All	0.01	1000	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
Model 1 All	0.01	1000	5.0025	3.6912	0.6909	0.6828	0.6909	0.6828	0.6903	0.6832
Model 1 selected	0.01	1000	2.297	2.2334	0.6902	0.6814	0.6902	0.6814	0.6896	0.6816
Model 2 Enhanced all	0.01	1000	27.099	28.2518	0.999	0.6346	0.999	0.6349	0.999	0.6661
Model 2 enhanced 2	0.001	50	1.4786	1.407	0.7411	0.6806	0.7411	0.6807	0.7501	0.6925
Model 2 enhanced and selected	0.001	50	1.4156	1.4354	0.7364	0.6826	0.7364	0.6826	0.7413	0.6904
Model 2 change learning rate and epochs and selected	0.0005	50	1.4849	1.4059	0.7101	0.6816	0.7101	0.6817	0.7165	0.6915
Model 3 deepwide all	0.001	50	3.6939	3.6335	0.7561	0.6805	0.7561	0.6804	0.7491	0.6738
Model 3 deepwide selected	0.001	50	3.6692	3.6169	0.7576	0.6806	0.7576	0.6807	0.7722	0.7029
Model 4 Hyper Parameter Tuning	Variable	20	Nan	Nan	0.7476	0.6761	0.7489	0.6843	0.7478	0.6772

Discussion of Results

In this study, several machine learning models were trained and evaluated to identify the best performing model. The models include logistic regression, k-nearest neighbors (KNN), support vector machines (SVM), decision trees, random forests, extra trees, bagging meta estimator, ADABoost SAMME, CATBoost, and ensemble learners (voting and stacking classifiers) and MLP models.

The new MLP model results presented show significant variation in the performance of these models in terms of accuracy, area under the curve (AUC), and F1 scores. In general, the enhanced MLP (model 2) and deep wide selected (model 3) have performed better compared to other models.

The Model 2 Enhanced exhibits very high training accuracy (0.9990) and F1 score (0.9990), but it performs poorly on the test dataset (accuracy: 0.6346, F1 score: 0.6661), indicating that the model is overfitting. Overfitting occurs when a model learns the training data too well and fails to generalize to unseen data.

On the other hand, some models like Model 1 and Model 2(change learning rate and epocs) display lower accuracy and F1 scores on both training and test sets. For example, Model 1 has a training accuracy of 0.6909 and F1 score of 0.6903, while the test accuracy is 0.6828 and F1 score is 0.6832. This is a sign of underfitting, which occurs when a model is not able to capture the underlying patterns in the data.

The enhanced MLP (Model 2), which has a training accuracy of 0.7411, test accuracy of 0.6806, training AUC of 0.7411, and test AUC of 0.6807, exhibits a more balanced performance across training and test datasets. Similarly, the F1 scores are 0.7501 and 0.6925 for training and test, respectively. This model has higher accuracy, AUC, and F1 scores compared to other models, indicating that it is able to generalize well to unseen data without overfitting or underfitting.

Another promising candidate is Model 3(Deep wide selected), with a training accuracy of 0.7576, test accuracy of 0.6806, training AUC of 0.7576, and test AUC of 0.6807. The F1 scores for training and test are 0.7722 and 0.7029, respectively. This model also demonstrates a good balance between avoiding overfitting and underfitting while maintaining good performance across different evaluation metrics.

In conclusion, the enhanced MLP (Model 2) and Model 3(Deep wide selected) appear to be the most promising candidates for this problem. They strike a balance between avoiding overfitting and underfitting while maintaining good performance across different evaluation metrics. Further tuning and optimization of these models could potentially lead to even better results.

Conclusion:

This project focused on predicting the probability of default for Home Credit clients using historical data, a vital aspect of informed lending decisions and minimizing unpaid loans. We hypothesized that machine learning models with custom features could accurately predict the risk of default.

In Phase 4, we expanded our analysis to include Multi-Layer Perceptron (MLP) models. The enhanced MLP (Model 2) with a training accuracy of 0.7411, test accuracy of 0.6806, and test F1 score of 0.6925 emerged as one of the most promising candidates. Model 3 (Deep wide selected) also showed strong performance, with a training accuracy of 0.7576, test accuracy of 0.6806, and test F1 score of 0.7029.

These results highlight the potential of Phase 4 models to help Home Credit make more accurate predictions on clients' likelihood to default, leading to better lending decisions and improved financial outcomes. Our work emphasizes the importance of feature engineering and hyperparameter tuning for optimizing model performance.

Future improvements can include experimenting with hyperparameters, regularization techniques, and Phase 4 model architectures. Enhancing feature selection, increasing dataset size, and utilizing advanced ensemble methods may boost the performance of enhanced MLP and Deep Wide Selected models, positively impacting lending decisions.

TODO: Utilizing the Featuretools library for automated feature engineering, as well as employing advanced models like TabNet,

LSTM, and Transfer Learning models beyond the ones available in PyTorch, to forecast loan repayment.

Please find the references below for your perusal:

Predict Loan Repayment with Automated Feature Engineering via Featuretools library: Github link: https://github.com/Featuretools/predict-loan-repayment/blob/master/Automated%20Loan%20Repayment.ipynb

A Guide to Automated Feature Engineering with Featuretools in Python: Link: https://www.analyticsvidhya.com/blog/2018/08/guide-automated-feature-engineering-featuretools-python/

Feature Engineering Paper: Link: https://dai.lids.mit.edu/wp-content/uploads/2017/10/DSAA_DSM_2015.pdf

Automated Categorical Data Analysis using CatBoost: Link: https://www.analyticsvidhya.com/blog/2017/08/catboost-automated-categorical-data/