Project done by : Janhavi Pramod Yarguddi

Contact no . : 9511794188

In [1]:

```
import pandas as pd
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from statsmodels.stats.outliers_influence import variance_inflation_factor
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import silhouette_score
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
Application_df = pd.read_csv("application_record (1).csv")
Record_df = pd.read_csv("credit_record.csv")
```

In [3]:

```
Application_df
```

Out[3]:

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN
0	5008804	M	Y	Y	0
1	5008805	M	Y	Y	0
2	5008806	M	Y	Y	0
3	5008808	F	N	Y	0
4	5008809	F	N	Y	0
...	...	...	...	...	...
438552	6840104	M	N	Y	0
438553	6840222	F	N	N	0
438554	6841878	F	N	N	0
438555	6842765	F	N	Y	0
438556	6842885	F	N	Y	0

438557 rows × 18 columns

In [4]:

```
Application_df["ID"].nunique()
```

Out[4]:

```
438510
```

In [5]:

```
Application_df.shape
```

Out[5]:

```
(438557, 18)
```

In [6]:

```
Application_df.drop_duplicates("ID",inplace=True,keep="last")
```

In [7]:

```
Application_df.shape
```

Out[7]:

```
(438510, 18)
```

In [8]:

```
Record_df
```

Out[8]:

	ID	MONTHS_BALANCE	STATUS
0	5001711	0	X
1	5001711	-1	0
2	5001711	-2	0
3	5001711	-3	0
4	5001712	0	C
...	...	...	...
1048570	5150487	-25	C
1048571	5150487	-26	C
1048572	5150487	-27	C
1048573	5150487	-28	C
1048574	5150487	-29	C

1048575 rows × 3 columns

In [9]:

```
len(set(Application_df["ID"]).intersection(set(Record_df["ID"])))  # Checking to see
```

Out[9]:

```
36457
```

In [10]:

```
Application_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 438510 entries, 0 to 438556
Data columns (total 18 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   ID                   438510 non-null  int64
 1   CODE_GENDER          438510 non-null  object
 2   FLAG_OWN_CAR         438510 non-null  object
 3   FLAG_OWN_REALTY      438510 non-null  object
 4   CNT_CHILDREN         438510 non-null  int64
 5   AMT_INCOME_TOTAL     438510 non-null  float64
 6   NAME_INCOME_TYPE     438510 non-null  object
 7   NAME_EDUCATION_TYPE  438510 non-null  object
 8   NAME_FAMILY_STATUS   438510 non-null  object
 9   NAME_HOUSING_TYPE    438510 non-null  object
 10  DAYS_BIRTH           438510 non-null  int64
 11  DAYS_EMPLOYED        438510 non-null  int64
 12  FLAG_MOBIL           438510 non-null  int64
 13  FLAG_WORK_PHONE      438510 non-null  int64
 14  FLAG_PHONE           438510 non-null  int64
 15  FLAG_EMAIL           438510 non-null  int64
 16  OCCUPATION_TYPE      304323 non-null  object
 17  CNT_FAM_MEMBERS      438510 non-null  float64
dtypes: float64(2), int64(8), object(8)
memory usage: 63.6+ MB
```

- Now here from the features given above and understanding the problem statement , features such as FLAG_MOBILE,FLAG_WORK_PHONE,FLAG_PHONE,FLAG_EMAIL can be dropped as these features do not affect the credit worthiness of any individual

```
Application_df.drop(["FLAG_MOBIL","FLAG_WORK_PHONE","FLAG_PHONE","FLAG_EMAIL"],axis=1,inplace=True)
Application_df.head()
```

In [11]:

```
Record_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 3 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   ID              1048575 non-null  int64
 1   MONTHS_BALANCE  1048575 non-null  int64
 2   STATUS          1048575 non-null  object
dtypes: int64(2), object(1)
memory usage: 24.0+ MB
```

In [12]:

```
status_count = Record_df.groupby("ID")["STATUS"].value_counts().unstack(fill_value=0
```

In [13]:

```
status_count["Average_Status"] = (status_count["0"]*-1+status_count["1"]*-2+status_c
```

In [14]:

```
status_count
```

Out[14]:

STATUS ID	0	1	2	3	4	5	C	X	Average_Status
5001711	3	0	0	0	0	0	0	1	-0.750000
5001712	10	0	0	0	0	0	9	0	-0.052632
5001713	0	0	0	0	0	0	0	22	0.000000
5001714	0	0	0	0	0	0	0	15	0.000000
5001715	0	0	0	0	0	0	0	60	0.000000
...	...	...	...	...	...	...	...	...	...
5150482	12	0	0	0	0	0	6	0	-0.333333
5150483	0	0	0	0	0	0	0	18	0.000000
5150484	12	0	0	0	0	0	1	0	-0.846154
5150485	2	0	0	0	0	0	0	0	-1.000000
5150487	0	0	0	0	0	0	30	0	1.000000

45985 rows × 9 columns

In [15]:

```
len(status_count[(status_count["Average_Status"]>0)])
```

Out[15]:

16633

In [16]:

```
 status_count.drop(["0","1","2","3","4","5","C","X"],axis=1,inplace=True)
```

- Here we have 2 datasets with a common feature "ID". Hence merge the datasets on the basis of "ID"

In [17]:

```
df = pd.merge(Application_df,status_count,on="ID",how="inner")
```

In [18]:

```
df.head()
```

Out[18]:

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_I
0	5008804	M	Y	Y	0	
1	5008805	M	Y	Y	0	
2	5008806	M	Y	Y	0	
3	5008808	F	N	Y	0	
4	5008809	F	N	Y	0	

In [19]:

```
df.shape
```

Out[19]:

```
(36457, 19)
```

In [20]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 36457 entries, 0 to 36456
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   36457 non-null  int64  
 1   CODE_GENDER          36457 non-null  object 
 2   FLAG_OWN_CAR         36457 non-null  object 
 3   FLAG_OWN_REALTY      36457 non-null  object 
 4   CNT_CHILDREN         36457 non-null  int64  
 5   AMT_INCOME_TOTAL     36457 non-null  float64
 6   NAME_INCOME_TYPE     36457 non-null  object 
 7   NAME_EDUCATION_TYPE  36457 non-null  object 
 8   NAME_FAMILY_STATUS   36457 non-null  object 
 9   NAME_HOUSING_TYPE    36457 non-null  object 
 10  DAYS_BIRTH           36457 non-null  int64  
 11  DAYS_EMPLOYED        36457 non-null  int64  
 12  FLAG_MOBIL           36457 non-null  int64  
 13  FLAG_WORK_PHONE      36457 non-null  int64  
 14  FLAG_PHONE           36457 non-null  int64  
 15  FLAG_EMAIL           36457 non-null  int64  
 16  OCCUPATION_TYPE      25134 non-null  object 
 17  CNT_FAM_MEMBERS      36457 non-null  float64
 18  Average_Status       36457 non-null  float64
dtypes: float64(3), int64(8), object(8)
memory usage: 5.6+ MB
```

- From the above output we understand that some of the values are in object datatype. But the ML model can be built on numerical values only. And hence we encode the values with object datatype

In [21]:

```
print(df["CODE_GENDER"].value_counts())
print("-"*50)
print(df["FLAG_OWN_CAR"].value_counts())
print("-"*50)
print(df["FLAG_OWN_REALTY"].value_counts())
print("-"*50)
print(df["NAME_INCOME_TYPE"].value_counts())
print("-"*50)
print(df["NAME_EDUCATION_TYPE"].value_counts())
print("-"*50)
print(df["NAME_FAMILY_STATUS"].value_counts())
print("-"*50)
print(df["NAME_HOUSING_TYPE"].value_counts())
print("-"*50)
print(df["OCCUPATION_TYPE"].value_counts())
```

```
F    24430
M    12027
Name: CODE_GENDER, dtype: int64
--------------------------------------------------
N    22614
Y    13843
Name: FLAG_OWN_CAR, dtype: int64
--------------------------------------------------
Y    24506
N    11951
Name: FLAG_OWN_REALTY, dtype: int64
--------------------------------------------------
Working                 18819
Commercial associate     8490
Pensioner                6152
State servant            2985
Student                    11
Name: NAME_INCOME_TYPE, dtype: int64
--------------------------------------------------
Secondary / secondary special    24777
Higher education                  9864
Incomplete higher                 1410
Lower secondary                    374
Academic degree                     32
Name: NAME_EDUCATION_TYPE, dtype: int64
--------------------------------------------------
Married                 25048
Single / not married     4829
Civil marriage           2945
Separated                2103
Widow                    1532
Name: NAME_FAMILY_STATUS, dtype: int64
--------------------------------------------------
House / apartment      32548
With parents            1776
Municipal apartment     1128
Rented apartment         575
Office apartment         262
Co-op apartment          168
Name: NAME_HOUSING_TYPE, dtype: int64
--------------------------------------------------
Laborers                 6211
Core staff               3591
Sales staff              3485
Managers                 3012
Drivers                  2138
High skill tech staff    1383
Accountants              1241
Medicine staff           1207
Cooking staff             655
Security staff            592
Cleaning staff            551
Private service staff     344
Low-skill Laborers        175
Waiters/barmen staff      174
Secretaries               151
HR staff                   85
Realty agents              79
IT staff                   60
Name: OCCUPATION_TYPE, dtype: int64
```

In [22]:

```
df["CODE_GENDER"] = df["CODE_GENDER"].replace({"F":1,"M":0})
df["FLAG_OWN_CAR"] = df["FLAG_OWN_CAR"].replace({"Y":1,"N":0})
df["FLAG_OWN_REALTY"] = df["FLAG_OWN_REALTY"].replace({"Y":1,"N":0})
df["NAME_INCOME_TYPE"] =pd.factorize(df.NAME_INCOME_TYPE)[0]
df["NAME_FAMILY_STATUS"] =pd.factorize(df.NAME_FAMILY_STATUS)[0]
df["NAME_HOUSING_TYPE"] =pd.factorize(df.NAME_HOUSING_TYPE)[0]
df["OCCUPATION_TYPE"] =pd.factorize(df.OCCUPATION_TYPE)[0]
```

Since we Education can be arranged in sequence so we use ordinal encoder for encoding the series "NAME_EDUCATION_TYPE"

In [23]:

```
from sklearn.preprocessing import OrdinalEncoder
```

In [24]:

```
ordinal_enc = OrdinalEncoder(categories=[["Secondary / secondary special","Higher ed
ordinal_enc.fit(df[["NAME_EDUCATION_TYPE"]])
df["NAME_EDUCATION_TYPE"] = ordinal_enc.transform(df[["NAME_EDUCATION_TYPE"]])
df["NAME_EDUCATION_TYPE"].unique()
```

Out[24]:

```
array([1., 0., 2., 3., 4.])
```

In [25]:

```
df["OCCUPATION_TYPE"].replace({-1:np.nan},inplace=True)
```

In [26]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 36457 entries, 0 to 36456
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   36457 non-null  int64  
 1   CODE_GENDER          36457 non-null  int64  
 2   FLAG_OWN_CAR         36457 non-null  int64  
 3   FLAG_OWN_REALTY      36457 non-null  int64  
 4   CNT_CHILDREN         36457 non-null  int64  
 5   AMT_INCOME_TOTAL     36457 non-null  float64
 6   NAME_INCOME_TYPE     36457 non-null  int64  
 7   NAME_EDUCATION_TYPE  36457 non-null  float64
 8   NAME_FAMILY_STATUS   36457 non-null  int64  
 9   NAME_HOUSING_TYPE    36457 non-null  int64  
 10  DAYS_BIRTH           36457 non-null  int64  
 11  DAYS_EMPLOYED        36457 non-null  int64  
 12  FLAG_MOBIL           36457 non-null  int64  
 13  FLAG_WORK_PHONE      36457 non-null  int64  
 14  FLAG_PHONE           36457 non-null  int64  
 15  FLAG_EMAIL           36457 non-null  int64  
 16  OCCUPATION_TYPE      25134 non-null  float64
 17  CNT_FAM_MEMBERS      36457 non-null  float64
 18  Average_Status       36457 non-null  float64
dtypes: float64(5), int64(14)
memory usage: 5.6 MB
```

In [27]:

```
df.shape
```

Out[27]:

```
(36457, 19)
```

In [28]:

```
df.head()
```

Out[28]:

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_I
0	5008804	0	1	1	0	
1	5008805	0	1	1	0	
2	5008806	0	1	1	0	
3	5008808	1	0	1	0	
4	5008809	1	0	1	0	

Now we have null values in the df["OCCUPATION_TYPE"] and hence we will use KNN imputer to fill in the null values.And now we have successfully changed the datatype of each series in the dataframe to numeric datatype

Since KNN imputer works on the basis of nan euclidean distance it is necessary to normalize the values i.e to bring all the values to a common scale

In [29]:

```
x = df.drop("ID",axis=1)
from sklearn.preprocessing import MinMaxScaler
normal_scalar = MinMaxScaler()
array  = normal_scalar.fit_transform(x)
normal_df = pd.DataFrame(array,columns=x.columns)
normal_df
```

Out[29]:

	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOI
0	0.0	1.0	1.0	0.0	
1	0.0	1.0	1.0	0.0	
2	0.0	1.0	1.0	0.0	
3	1.0	0.0	1.0	0.0	
4	1.0	0.0	1.0	0.0	
...	...	...	...	...	
36452	0.0	1.0	1.0	0.0	
36453	1.0	0.0	1.0	0.0	
36454	1.0	0.0	1.0	0.0	
36455	1.0	0.0	1.0	0.0	
36456	0.0	0.0	1.0	0.0	

36457 rows × 18 columns

Filling missing values

In [30]:

```
from sklearn.impute import KNNImputer
knn_imputer = KNNImputer(n_neighbors=10)
array = knn_imputer.fit_transform(normal_df)
df1 = pd.DataFrame(array,columns=normal_df.columns)
df1
```

Out[30]:

	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCON
0	0.0	1.0	1.0	0.0	
1	0.0	1.0	1.0	0.0	
2	0.0	1.0	1.0	0.0	
3	1.0	0.0	1.0	0.0	
4	1.0	0.0	1.0	0.0	
...	...	...	...	...	
36452	0.0	1.0	1.0	0.0	
36453	1.0	0.0	1.0	0.0	
36454	1.0	0.0	1.0	0.0	
36455	1.0	0.0	1.0	0.0	
36456	0.0	0.0	1.0	0.0	

36457 rows × 18 columns

By keeping on trying and changing the kneighbours we will keep on getting better results. It is just the trial and error method that can be applied here to get better and better results.

In [31]:

```
df1["ID"] = df["ID"]
```

In [32]:

```
df1.head()
```

Out[32]:

	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_T
0	0.0	1.0	1.0	0.0	0.2
1	0.0	1.0	1.0	0.0	0.2
2	0.0	1.0	1.0	0.0	0.0
3	1.0	0.0	1.0	0.0	0.1
4	1.0	0.0	1.0	0.0	0.1

Use of ELBOW method to find out the required no of clusters

In [33]:

```
wcss_lst = []
k_values = range(1,5)
for i in k_values:
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(df1)
    wcss = kmeans.inertia_
    wcss_lst.append(wcss)
    print("WCSS",wcss,i)
```

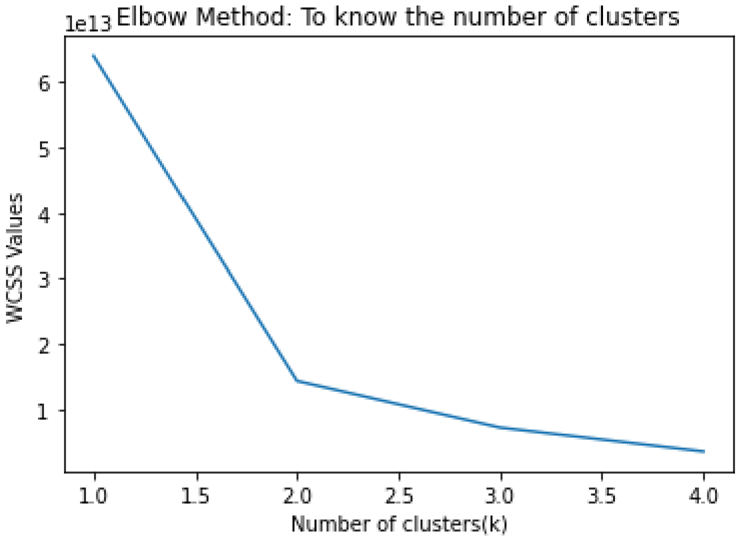
```
WCSS 63926900854690.17 1
WCSS 14382282678440.19 2
WCSS 7268517637918.215 3
WCSS 3623818974645.6187 4
```

In [34]:

```
plt.plot(k_values,wcss_lst)
plt.xlabel("Number of clusters(k)")
plt.ylabel("WCSS Values")
plt.title("Elbow Method: To know the number of clusters")
```

Out[34]:

```
Text(0.5, 1.0, 'Elbow Method: To know the number of clusters')
```



MODEL TRAINING

In [35]:

```
km_obj = KMeans(n_clusters=2) # k=8 default
km_obj.fit(df1)
y_pred = km_obj.fit_predict(df1)
df1["Target"] = y_pred
```

In [36]:

```
score = silhouette_score(df1,y_pred)
score
```

Out[36]:

```
0.6653479837032964
```

In [37]:

```
df1.head(10)
```

Out[37]:

	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_T
0	0.0	1.0	1.0	0.0	0.2
1	0.0	1.0	1.0	0.0	0.2
2	0.0	1.0	1.0	0.0	0.0
3	1.0	0.0	1.0	0.0	0.1
4	1.0	0.0	1.0	0.0	0.1
5	1.0	0.0	1.0	0.0	0.1
6	1.0	0.0	1.0	0.0	0.1
7	1.0	0.0	1.0	0.0	0.1
8	1.0	0.0	1.0	0.0	0.1
9	1.0	0.0	1.0	0.0	0.1

In [38]:

```
df1["Target"].value_counts()
```

Out[38]:

```
0    18314
1    18143
Name: Target, dtype: int64
```

In [39]:

```
km_obj = KMeans(n_clusters=3) # k=8 default
km_obj.fit(df1)
y_pred = km_obj.fit_predict(df1)
df1["Target"] = y_pred
```

In [40]:

```
df1["Target"].value_counts()
```

Out[40]:

```
2    13760
1    11870
0    10827
Name: Target, dtype: int64
```

Now we check the silhouette score to know how well the data points fit to the respective clusters.

In [41]:

```
score = silhouette_score(df1,y_pred)
score
```

Out[41]:

0.5909416434253694

We have build the model without analysing the features (feature selection). Now we will look at the outliers, deal with it if any. Remove or add features based on the requirement and on correlation or vif values

In [42]:

```
knn_imputer = KNNImputer(n_neighbors=15)
array = knn_imputer.fit_transform(normal_df)
df2 = pd.DataFrame(array,columns=normal_df.columns)
df2
```

Out[42]:

	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOI
0	0.0	1.0	1.0	0.0	
1	0.0	1.0	1.0	0.0	
2	0.0	1.0	1.0	0.0	
3	1.0	0.0	1.0	0.0	
4	1.0	0.0	1.0	0.0	
...	...	...	...	...	
36452	0.0	1.0	1.0	0.0	
36453	1.0	0.0	1.0	0.0	
36454	1.0	0.0	1.0	0.0	
36455	1.0	0.0	1.0	0.0	
36456	0.0	0.0	1.0	0.0	

36457 rows × 18 columns

In [43]:

```
df2["ID"] = df["ID"]
```

Outliers detection

In [44]:

```
df2.columns
```

Out[44]:

```
Index(['CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN',
       'AMT_INCOME_TOTAL', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
       'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'DAYS_BIRTH',
       'DAYS_EMPLOYED', 'FLAG_MOBIL', 'FLAG_WORK_PHONE', 'FLAG_PHONE',
       'FLAG_EMAIL', 'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS', 'Average_Statu
s',
       'ID'],
      dtype='object')
```

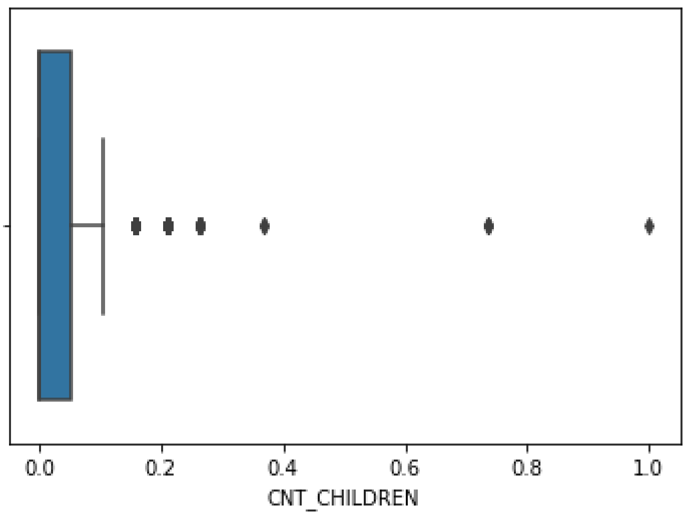
```
def get_value_count(Data):
    for i in Data.columns:
        print(Data[i].value_counts())
        print("-"*50)
get_value_count(df)
```

In [45]:

```
sns.boxplot(df2["CNT_CHILDREN"])
```

Out[45]:

```
<AxesSubplot:xlabel='CNT_CHILDREN'>
```

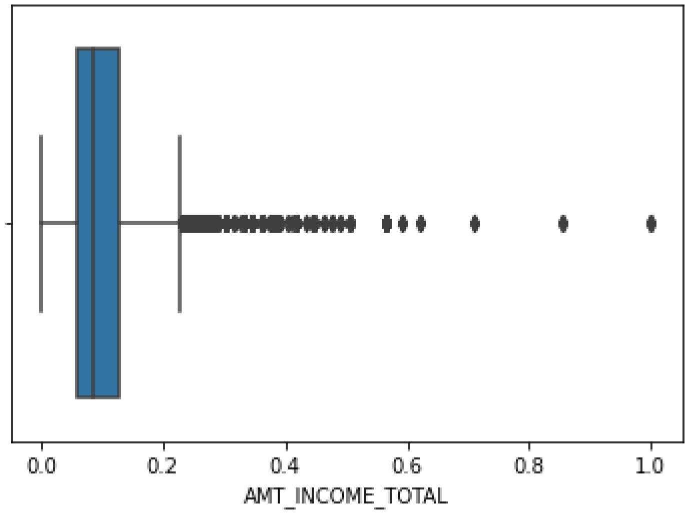


In [46]:

```
sns.boxplot(df2["AMT_INCOME_TOTAL"])
```

Out[46]:

```
<AxesSubplot:xlabel='AMT_INCOME_TOTAL'>
```



Here we have too many outliers present in these 2 features. But these features are important ones and indivisually affect the capability of an individual to pay off debts. So in my opinion it is better if we keep these outliers.

Feature Selection

In [47]:

```
df2.corr()
```

Out[47]:

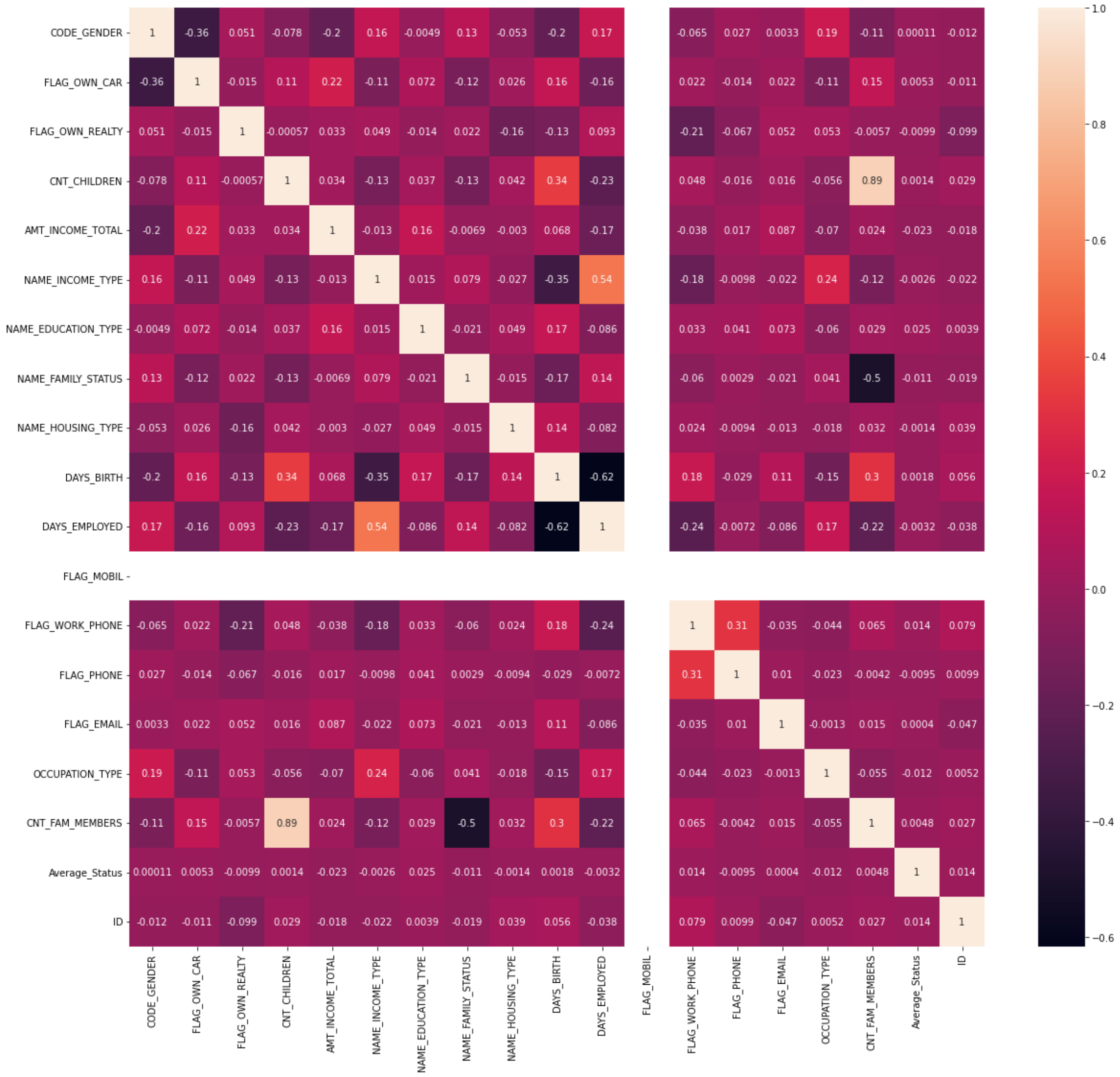
	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHIL
CODE_GENDER	1.000000	-0.361379	0.050758	-0.(
FLAG_OWN_CAR	-0.361379	1.000000	-0.015185	0.'
FLAG_OWN_REALTY	0.050758	-0.015185	1.000000	-0.(
CNT_CHILDREN	-0.077690	0.105839	-0.000575	1.(
AMT_INCOME_TOTAL	-0.197805	0.215506	0.032719	0.(
NAME_INCOME_TYPE	0.158594	-0.110925	0.049448	-0.'
NAME_EDUCATION_TYPE	-0.004908	0.072104	-0.014143	0.(
NAME_FAMILY_STATUS	0.125867	-0.119062	0.022472	-0.'
NAME_HOUSING_TYPE	-0.053499	0.025786	-0.163836	0.(
DAYS_BIRTH	-0.202352	0.157144	-0.129838	0.:
DAYS_EMPLOYED	0.173434	-0.156452	0.093006	-0.:
FLAG_MOBIL	NaN	NaN	NaN	
FLAG_WORK_PHONE	-0.064994	0.021644	-0.207732	0.(
FLAG_PHONE	0.026833	-0.014019	-0.066601	-0.(
FLAG_EMAIL	0.003284	0.021750	0.052194	0.(
OCCUPATION_TYPE	0.187171	-0.110235	0.052728	-0.(
CNT_FAM_MEMBERS	-0.110782	0.151814	-0.005723	0.:
Average_Status	0.000114	0.005253	-0.009948	0.(
ID	-0.012022	-0.011163	-0.098851	0.(

In [48]:

```
plt.figure(figsize=(20,18))
sns.heatmap(df2.corr(),annot=True)
```

Out[48]:

<AxesSubplot:>



In [49]:

```
1 df2.corr().loc["AMT_INCOME_TOTAL"].sort_values(ascending=False)
```

Out[49]:

```
AMT_INCOME_TOTAL       1.000000
FLAG_OWN_CAR           0.215506
NAME_EDUCATION_TYPE    0.158380
FLAG_EMAIL             0.086681
DAYS_BIRTH             0.067908
CNT_CHILDREN           0.033691
FLAG_OWN_REALTY        0.032719
CNT_FAM_MEMBERS        0.023750
FLAG_PHONE             0.017245
NAME_HOUSING_TYPE     -0.003048
NAME_FAMILY_STATUS    -0.006935
NAME_INCOME_TYPE      -0.012952
ID                    -0.017667
Average_Status        -0.022594
FLAG_WORK_PHONE       -0.037746
OCCUPATION_TYPE       -0.069749
DAYS_EMPLOYED         -0.168611
CODE_GENDER           -0.197805
FLAG_MOBIL                  NaN
Name: AMT_INCOME_TOTAL, dtype: float64
```

In [50]:

```
1 df2.corr().loc["Average_Status"].sort_values(ascending=False)
```

Out[50]:

```
Average_Status         1.000000
NAME_EDUCATION_TYPE    0.024622
ID                     0.013994
FLAG_WORK_PHONE        0.013772
FLAG_OWN_CAR           0.005253
CNT_FAM_MEMBERS        0.004814
DAYS_BIRTH             0.001806
CNT_CHILDREN           0.001417
FLAG_EMAIL             0.000398
CODE_GENDER            0.000114
NAME_HOUSING_TYPE     -0.001438
NAME_INCOME_TYPE      -0.002555
DAYS_EMPLOYED         -0.003228
FLAG_PHONE            -0.009510
FLAG_OWN_REALTY       -0.009948
NAME_FAMILY_STATUS    -0.010597
OCCUPATION_TYPE       -0.012282
AMT_INCOME_TOTAL      -0.022594
FLAG_MOBIL                  NaN
Name: Average_Status, dtype: float64
```

In [51]:

```
vif = pd.DataFrame()
vif["Feature"] = df2.columns
vif["VIF_Values"] = [variance_inflation_factor(df2.to_numpy(),i) for i in range(df2.
vif
```

Out[51]:

	Feature	VIF_Values
0	CODE_GENDER	3.771607
1	FLAG_OWN_CAR	1.957894
2	FLAG_OWN_REALTY	3.311363
3	CNT_CHILDREN	17.991634
4	AMT_INCOME_TOTAL	3.974367
5	NAME_INCOME_TYPE	2.496139
6	NAME_EDUCATION_TYPE	1.478556
7	NAME_FAMILY_STATUS	11.888564
8	NAME_HOUSING_TYPE	4.782598
9	DAYS_BIRTH	11.076880
10	DAYS_EMPLOYED	2.739023
11	FLAG_MOBIL	NaN
12	FLAG_WORK_PHONE	1.610096
13	FLAG_PHONE	1.597215
14	FLAG_EMAIL	1.131784
15	OCCUPATION_TYPE	4.077549
16	CNT_FAM_MEMBERS	46.055052
17	Average_Status	71.327042
18	ID	144.021412

- From the above analysis, VIF values for features CNT_CHILDREN,NAME_FAMILY_STATUS,DAYS_BIRTH,FLAG_MOBIL,CNT_FAM_MEMBERS,ID is either greater than 10 or not available. And hence we can drop these features. As VIF rightly indicates multicollinearity between the features. We here are not considering the feature "Average_status" since we know that this feature has come into picture taking into consideration all the other features, it is supposed to have a higher VIF.

In [52]:

```
df2.drop(["CNT_CHILDREN","FLAG_MOBIL","ID"],axis=1,inplace=True)
```

Use of ELBOW method to find out the required no of clusters

In [53]:

```
wcss_lst = []
k_values = range(1,5)
for i in k_values:
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(df2)
    wcss = kmeans.inertia_
    wcss_lst.append(wcss)
    print("WCSS",wcss,i)
```

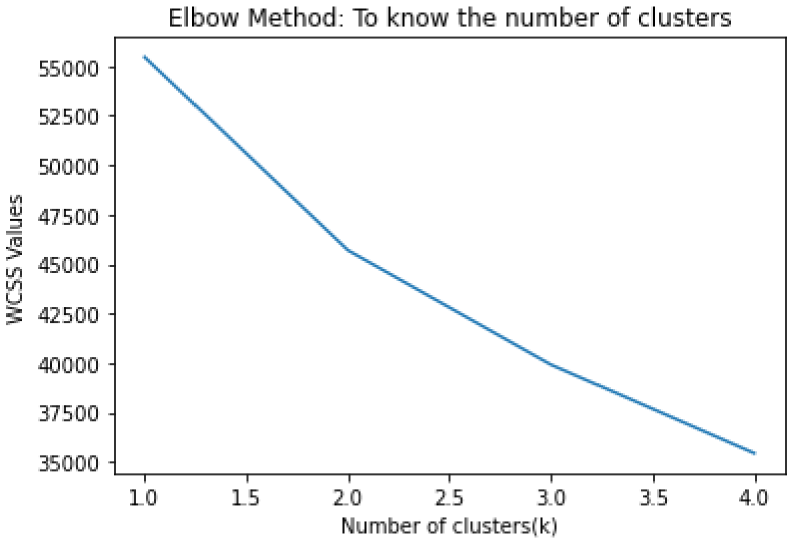
```
WCSS 55458.80291958559 1
WCSS 45692.75199988235 2
WCSS 39914.176593071046 3
WCSS 35438.66275679613 4
```

In [54]:

```
plt.plot(k_values,wcss_lst)
plt.xlabel("Number of clusters(k)")
plt.ylabel("WCSS Values")
plt.title("Elbow Method: To know the number of clusters")
```

Out[54]:

```
Text(0.5, 1.0, 'Elbow Method: To know the number of clusters')
```



- There ia slight bend at k=2 i.e number of clusters=2

MODEL TRAINING

In [55]:

```
KM_obj = KMeans(n_clusters=3) # k=8 default
KM_obj.fit(df2)
y_pred = KM_obj.fit_predict(df2)
df1["Target"] = y_pred
```

In [56]:

```
score = silhouette_score(df2,y_pred)
score
```

Out[56]:

```
0.1956739020205943
```

In [57]:

```
KM_obj = KMeans(n_clusters=2) # k=8 default
KM_obj.fit(df2)
y_pred = KM_obj.fit_predict(df2)
df1["Target"] = y_pred
```

In [58]:

```
score = silhouette_score(df2,y_pred)
score
```

Out[58]:

```
0.18255350709499665
```

Type *Markdown* and LaTeX: α^2

Silhouette score actually ranges from -1 to +1. It actually indicates how properly the clusters have been formed. After applying feature selection techniques like vif (which is actually used and applicable mostly in regression type of problems) and removing the features the silhouette score actually decreased. Hence we can say that all the above features are important in deciding whether a customer is a good customer or a bad customer i.e whether a customer will default or not.

Hence here the model km_obj with cluster=2 is the best model with a silhouette score of 0.6653