Self Case Study - 1

Customer Relationship Prediction - Churn

In [ ]:

```
!pip install -U scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (0.22.2.post1)
Collecting scikit-learn
  Downloading scikit_learn-0.24.2-cp37-cp37m-manylinux2010_x86_64.whl (22.3 MB)
     |████████████████████████████████| 22.3 MB 1.6 MB/s
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-lear
n) (1.0.1)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/dist-packages (from scikit-lea
rn) (1.4.1)
Collecting threadpoolctl>=2.0.0
  Downloading threadpoolctl-2.2.0-py3-none-any.whl (12 kB)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages (from scikit-lea
rn) (1.19.5)
Installing collected packages: threadpoolctl, scikit-learn
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 0.22.2.post1
    Uninstalling scikit-learn-0.22.2.post1:
      Successfully uninstalled scikit-learn-0.22.2.post1
Successfully installed scikit-learn-0.24.2 threadpoolctl-2.2.0
```

In [ ]:

```
!pip install dython
```

```
Collecting dython
  Downloading dython-0.6.7-py3-none-any.whl (19 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from dython) (1.19.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (from dython) (3.2.
2)
Requirement already satisfied: seaborn in /usr/local/lib/python3.7/dist-packages (from dython) (0.11.1)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from dython) (1.4.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (from dython) (0.
24.2)
Requirement already satisfied: pandas>=0.23.4 in /usr/local/lib/python3.7/dist-packages (from dython) (
1.1.5)
Collecting scikit-plot>=0.3.7
  Downloading scikit_plot-0.3.7-py3-none-any.whl (33 kB)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.2
3.4->dython) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from p
andas>=0.23.4->dython) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil
>=2.7.3->pandas>=0.23.4->dython) (1.15.0)
Requirement already satisfied: joblib>=0.10 in /usr/local/lib/python3.7/dist-packages (from scikit-plot
>=0.3.7->dython) (1.0.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib-
>dython) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dis
t-packages (from matplotlib->dython) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplo
tlib->dython) (1.3.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from sci
kit-learn->dython) (2.2.0)
Installing collected packages: scikit-plot, dython
Successfully installed dython-0.6.7 scikit-plot-0.3.7
```

In [ ]:

```
!pip install fast-ml
```

```
Collecting fast-ml
  Downloading fast_ml-3.68-py3-none-any.whl (42 kB)
     |████████████████████████████████| 42 kB 716 kB/s
Installing collected packages: fast-ml
Successfully installed fast-ml-3.68
```

In [ ]:

```python
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.preprocessing import OrdinalEncoder
from sklearn.manifold import TSNE
from sklearn.preprocessing import PolynomialFeatures

import missingno as msno
from dython import nominal

import pickle

from sklearn.cluster import DBSCAN, KMeans

from fast_ml.utilities import display_all
from fast_ml.feature_selection import get_duplicate_features

from sklearn.model_selection import train_test_split

from prettytable import PrettyTable

%matplotlib inline
```

Loading data

In [ ]:

```python
data = pd.read_csv('/content/drive/MyDrive/Case Study 1/Data/EDA/orange_small_train.data', sep = '\t')
```

In [ ]:

```python
data.shape
```

Out[ ]:

```
(50000, 230)
```

In [ ]:

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Columns: 230 entries, Var1 to Var230
dtypes: float64(191), int64(1), object(38)
memory usage: 87.7+ MB
```

There are total of 50k datapoints and each datapoint has 230 features.

In [ ]:

```python
data.head()
```

Out[ ]:

	Var1	Var2	Var3	Var4	Var5	Var6	Var7	Var8	Var9	Var10	Var11	Var12	Var13	Var14	Var15	Var16	Var17	Var18
0	NaN	NaN	NaN	NaN	NaN	1526.0	7.0	NaN	NaN	NaN	NaN	NaN	184.0	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	525.0	0.0	NaN	NaN	NaN	NaN	NaN	0.0	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	5236.0	7.0	NaN	NaN	NaN	NaN	NaN	904.0	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	NaN	NaN	NaN	NaN	0.0	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	1029.0	7.0	NaN	NaN	NaN	NaN	NaN	3216.0	NaN	NaN	NaN	NaN	NaN

5 rows × 230 columns

In [ ]:

```
data.describe()
```

Out[ ]:

	Var1	Var2	Var3	Var4	Var5	Var6	Var7	Var8	
count	702.000000	1241.000000	1240.000000	1579.000000	1.487000e+03	44471.000000	44461.000000	0.0	702.000
mean	11.487179	0.004029	425.298387	0.125396	2.387933e+05	1326.437116	6.809496	NaN	48.1452
std	40.709951	0.141933	4270.193518	1.275481	6.441259e+05	2685.693668	6.326053	NaN	154.777
min	0.000000	0.000000	0.000000	0.000000	0.000000e+00	0.000000	0.000000	NaN	0.00000
25%	0.000000	0.000000	0.000000	0.000000	0.000000e+00	518.000000	0.000000	NaN	4.00000
50%	0.000000	0.000000	0.000000	0.000000	0.000000e+00	861.000000	7.000000	NaN	20.0000
75%	16.000000	0.000000	0.000000	0.000000	1.187425e+05	1428.000000	7.000000	NaN	46.0000
max	680.000000	5.000000	130668.000000	27.000000	6.048550e+06	131761.000000	140.000000	NaN	2300.00

8 rows × 192 columns

In [ ]:

```
churn_labels = pd.read_csv('/content/drive/MyDrive/Case Study 1/Data/EDA/orange_small_train_churn.label
s', header = None,names = ['Churn'])
```

In [ ]:

```
churn_labels.head()
```

Out[ ]:

	Churn
0	-1
1	1
2	-1
3	-1
4	-1

In [ ]:

```
churn_labels['Churn'] = churn_labels.Churn.apply(lambda x: 0 if (x == -1) else x)
```

In [ ]:

```
churn_labels.shape
```

Out[ ]:

```
(50000, 1)
```

In [ ]:

```
churn_labels.head()
```

Out[ ]:

	Churn
0	0
1	1
2	0
3	0
4	0

Splitting data into train and test before data analysis

In [ ]:

```
X_train_churn, X_test_churn, y_train_churn, y_test_churn = train_test_split(data,churn_labels, test_siz
e = 0.2, stratify = churn_labels)
```

EDA

Class distribution

In [ ]:

```
def plot_class_dist(x, data):
    sns.countplot(x = x,data =  data)
    plt.title('{} class label value counts'.format(x))
    plt.show()
```

In [ ]:

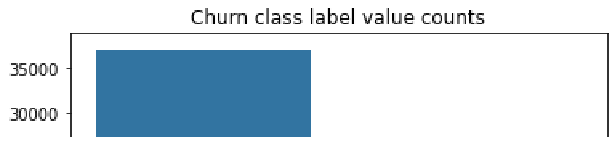
```
y_train_churn.value_counts()
```

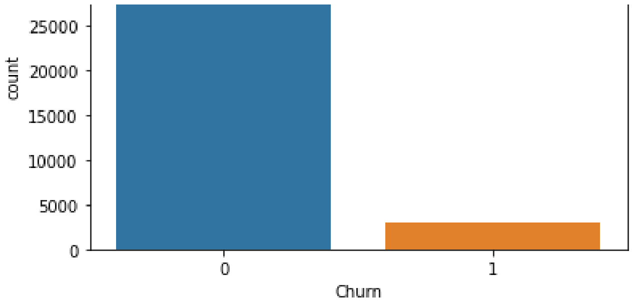
Out[ ]:

```
Churn
0        37062
1         2938
dtype: int64
```

In [ ]:

```
plot_class_dist('Churn', y_train_churn)
```





Observation:

- Data w.r.t churn label is highly imbalanced.

Counting total NaNs for each feature

In [ ]:

```
print('Number of features which only have NaNs present: ',(X_train_churn.isna().sum() == X_train_churn.
shape[0]).sum())
```

```
Number of features which only have NaNs present:  18
```

In [ ]:

```
all_nan_columns = np.array(X_train_churn.columns[X_train_churn.isna().sum() == X_train_churn.shape[0]])
```

In [ ]:

```
print('Number of features which do not countain NaNs:',(X_train_churn.notna().sum() == X_train_churn.sh
ape[0]).sum())
```

```
Number of features which do not countain NaNs: 19
```

In [ ]:

```
not_nan_columns = np.array(X_train_churn.columns[X_train_churn.notna().sum() == X_train_churn.shape[0]]
)
```

In [ ]:

```
not_nan_columns
```

Out[ ]:

```
array(['Var57', 'Var73', 'Var113', 'Var193', 'Var195', 'Var196', 'Var198',
       'Var204', 'Var207', 'Var210', 'Var211', 'Var212', 'Var216',
       'Var220', 'Var221', 'Var222', 'Var226', 'Var227', 'Var228'],
      dtype=object)
```

In [ ]:

```
X_train_churn.dtypes[not_nan_columns]
```

Out[ ]:

```
Var57      float64
Var73        int64
Var113     float64
Var193      object
Var195      object
Var196      object
Var198      object
```

```
Var204      object
Var207      object
Var210      object
Var211      object
Var212      object
Var216      object
Var220      object
Var221      object
Var222      object
Var226      object
Var227      object
Var228      object
dtype: object
```

Observation:

    - Out of 19 columns which do not have any missing data, 3 are numerical and 16 are categorical.

In [ ]:

```python
#https://stackoverflow.com/questions/26266362/how-to-count-the-nan-values-in-a-column-in-pandas-datafra
me
nan_count_array = []
for i in X_train_churn.columns:
    nan_count = X_train_churn[i].isna().sum()
    nan_count_array.append(nan_count)
```

In [ ]:

```python
#to-do: tabular form
x = PrettyTable()
x.add_column('Features', list(X_train_churn.columns))
x.add_column('Number of NaNs',nan_count_array)
```

In [ ]:

```python
print(x)
```

```
+----------+----------------+
| Features | Number of NaNs |
+----------+----------------+
|   Var1   |     39436      |
|   Var2   |     38983      |
|   Var3   |     38983      |
|   Var4   |     38715      |
|   Var5   |     38790      |
|   Var6   |      4500      |
|   Var7   |      4493      |
|   Var8   |     40000      |
|   Var9   |     39436      |
|  Var10   |     38790      |
|  Var11   |     38983      |
|  Var12   |     39552      |
|  Var13   |      4493      |
|  Var14   |     38983      |
|  Var15   |     40000      |
|  Var16   |     38790      |
|  Var17   |     38715      |
|  Var18   |     38715      |
|  Var19   |     38715      |
|  Var20   |     40000      |
|  Var21   |      4500      |
|  Var22   |      4076      |
|  Var23   |     38790      |
|  Var24   |      5848      |
|  Var25   |      4076      |
|  Var26   |     38790      |
|  Var27   |     38790      |
|  Var28   |      4078      |
|  Var29   |     39436      |
```

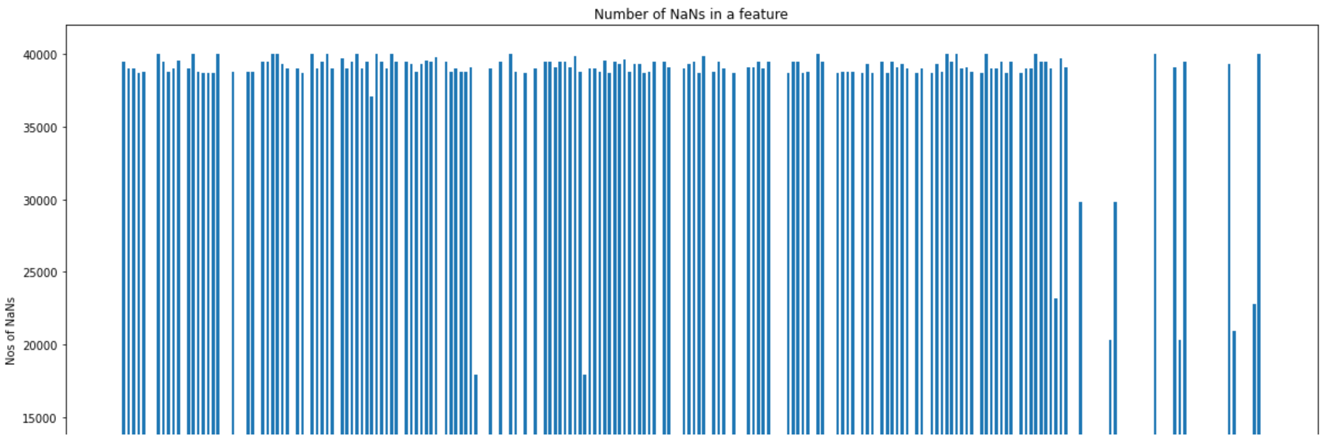
Var30	39436
Var31	40000
Var32	40000
Var33	39301
Var34	38983
Var35	4076
Var36	38983
Var37	38715
Var38	4076
Var39	40000
Var40	38983
Var41	39436
Var42	40000
Var43	38983
Var44	4076
Var45	39711
Var46	38983
Var47	39436
Var48	40000
Var49	38983
Var50	39436
Var51	37052
Var52	40000
Var53	39436
Var54	38983
Var55	40000
Var56	39465
Var57	0
Var58	39436
Var59	39322
Var60	38790
Var61	39301
Var62	39552
Var63	39442
Var64	39806
Var65	4493
Var66	39442
Var67	38790
Var68	38983
Var69	38790
Var70	38790
Var71	39077
Var72	17931
Var73	0
Var74	4493
Var75	38983
Var76	4076
Var77	39436
Var78	4076
Var79	40000
Var80	38790
Var81	4500
Var82	38715
Var83	4076
Var84	38983
Var85	4076
Var86	39436
Var87	39436
Var88	39116
Var89	39465
Var90	39436
Var91	39077
Var92	39850
Var93	38790
Var94	17931
Var95	38983
Var96	38983
Var97	38790
Var98	39552
Var99	38715
Var100	39436
Var101	39292
Var102	39633
Var103	38790
Var104	39322
Var105	39322
Var106	38715

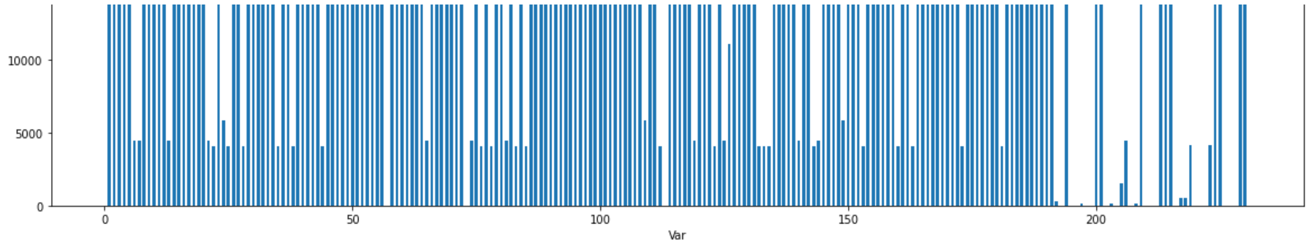
Var107	38790
Var108	39436
Var109	5848
Var110	39436
Var111	39077
Var112	4076
Var113	0
Var114	38983
Var115	39322
Var116	39436
Var117	38715
Var118	39850
Var119	4500
Var120	38790
Var121	39436
Var122	38983
Var123	4076
Var124	38715
Var125	4493
Var126	11135
Var127	39116
Var128	39116
Var129	39436
Var130	38983
Var131	39436
Var132	4076
Var133	4076
Var134	4076
Var135	38715
Var136	39442
Var137	39436
Var138	38715
Var139	38790
Var140	4493
Var141	40000
Var142	39436
Var143	4076
Var144	4500
Var145	38715
Var146	38790
Var147	38790
Var148	38790
Var149	5848
Var150	38715
Var151	39301
Var152	38715
Var153	4076
Var154	39436
Var155	38715
Var156	39442
Var157	39077
Var158	39292
Var159	38983
Var160	4076
Var161	38715
Var162	38983
Var163	4076
Var164	38715
Var165	39292
Var166	38790
Var167	40000
Var168	39436
Var169	40000
Var170	38983
Var171	39116
Var172	38790
Var173	4076
Var174	38715
Var175	40000
Var176	38983
Var177	38983
Var178	39465
Var179	38715
Var180	39436
Var181	4076
Var182	38715
Var183	38983

```
|  Var184  |     38983      |
|  Var185  |     40000      |
|  Var186  |     39436      |
|  Var187  |     39436      |
|  Var188  |     38983      |
|  Var189  |     23172      |
|  Var190  |     39732      |
|  Var191  |     39116      |
|  Var192  |      289       |
|  Var193  |       0        |
|  Var194  |     29802      |
|  Var195  |       0        |
|  Var196  |       0        |
|  Var197  |      114       |
|  Var198  |       0        |
|  Var199  |       4        |
|  Var200  |     20306      |
|  Var201  |     29803      |
|  Var202  |       1        |
|  Var203  |      114       |
|  Var204  |       0        |
|  Var205  |      1555      |
|  Var206  |      4500      |
|  Var207  |       0        |
|  Var208  |      114       |
|  Var209  |     40000      |
|  Var210  |       0        |
|  Var211  |       0        |
|  Var212  |       0        |
|  Var213  |     39077      |
|  Var214  |     20306      |
|  Var215  |     39442      |
|  Var216  |       0        |
|  Var217  |      564       |
|  Var218  |      564       |
|  Var219  |      4173      |
|  Var220  |       0        |
|  Var221  |       0        |
|  Var222  |       0        |
|  Var223  |      4173      |
|  Var224  |     39322      |
|  Var225  |     20907      |
|  Var226  |       0        |
|  Var227  |       0        |
|  Var228  |       0        |
|  Var229  |     22741      |
|  Var230  |     40000      |
+----------+----------------+
```

In [ ]:

```
fig = plt.figure(figsize = (20, 10))
plt.bar(range(1,X_train_churn.shape[1]+1), height = nan_count_array, width = 0.6,data = nan_count_array
)
plt.xlabel('Var')
plt.ylabel('Nos of NaNs')
plt.title('Number of NaNs in a feature')
plt.show()
```





Observation:

- Most of the features have high count of NaNs (near to 50k)
- Few features have NaN count under 10k
- Only a handful of features (belonging to categorical) have low or none count of NaNs
- There are 18 columns with only NaN value present

Unique value counts for Categorical features

- Categorical features ranges from Var191 to Var230
- We'll see the the unique values that each feature holds. Based on the count, we'll decide which categorical encoding to choose.
- If the value count per feature is high, then choosing OHE will result in highly sparse and large vectors.

In [ ]:

```
X_train_churn.iloc[:,190:].head()
```

Out[ ]:

	Var191	Var192	Var193	Var194	Var195	Var196	Var197	Var198	Var199	V
13282	NaN	oUPBcmzkzH	5QKljwyXr4MCZTEp7uAkS8PtBLcn	SEuy	taul	1K8T	lK27	fhk21Ss	r83_sZi	Xu
44013	NaN	52Iq9ayE15	RO12	NaN	taul	1K8T	z32l	UsSOoyT	nQUq7hGe64	Na
30999	NaN	a4vPe2fHUn	2Knk1KF	SEuy	taul	1K8T	TyGl	THRJJYr	r83_sZi	So
7521	NaN	EsYq9aX0Db	RO12	NaN	taul	1K8T	L80O	8K14q6X	PaagavI	Na
17794	NaN	1YVvyx7IEC	RO12	NaN	taul	1K8T	FgS1	oKsWccX	Gai9IEF2Fr	Na

In [ ]:

```
X_train_churn.iloc[:,190:].nunique(axis = 0,dropna = False)
```

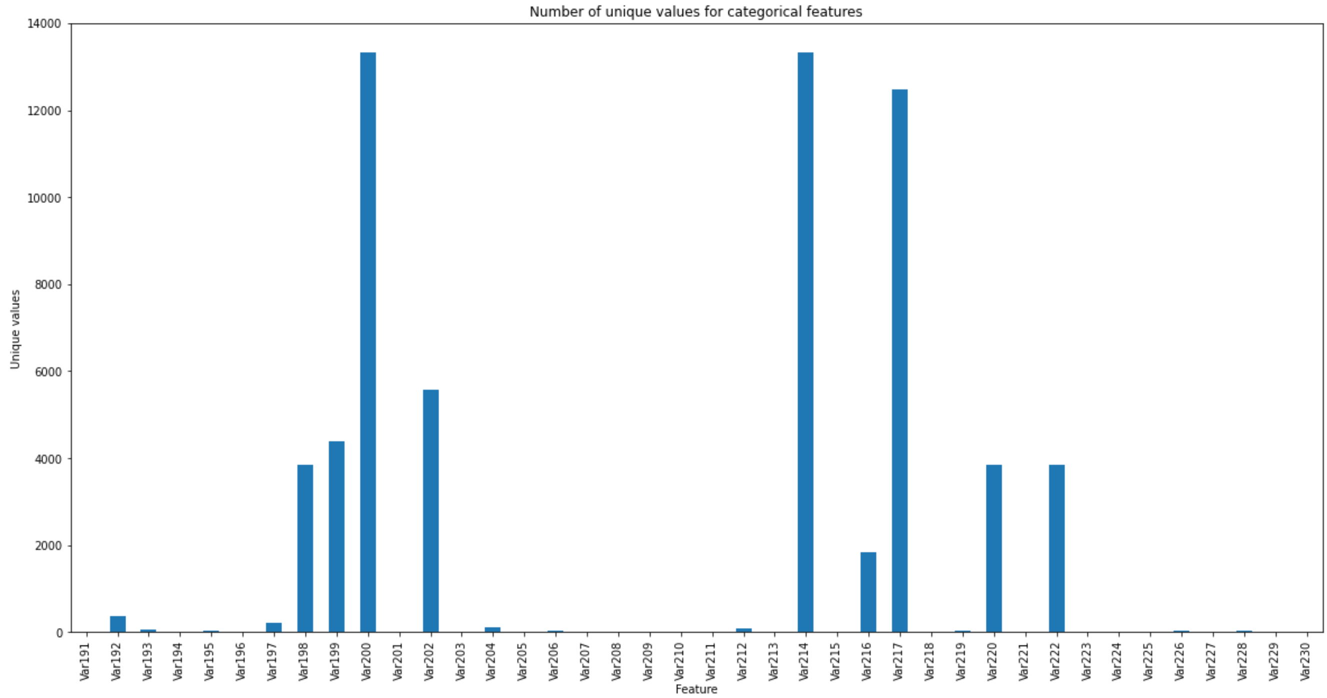
Out[ ]:

```
Var191        2
Var192      357
Var193       51
Var194        4
Var195       22
Var196        4
Var197      218
Var198     3844
Var199     4395
Var200    13338
Var201        3
Var202     5561
Var203        6
Var204      100
Var205        4
Var206       22
Var207       14
Var208        3
Var209        1
Var210        6
Var211        2
Var212       77
```

```
Var213        2
Var214    13338
Var215        2
Var216     1838
Var217    12479
Var218        3
Var219       23
Var220     3844
Var221        7
Var222     3844
Var223        5
Var224        2
Var225        4
Var226       23
Var227        7
Var228       30
Var229        5
Var230        1
dtype: int64
```

In [ ]:

```python
#https://www.geeksforgeeks.org/how-to-count-distinct-values-of-a-pandas-dataframe-column/
fig = plt.figure(figsize = (20, 10))
X_train_churn.iloc[:,190:].nunique(axis = 0,dropna = False).plot(kind = 'bar')
plt.title('Number of unique values for categorical features')
plt.xlabel('Feature')
plt.ylabel('Unique values')
plt.show()
```



Observation:

- 20 out of 40 feature have unique value count under 10.
- 9 features have unique value count which spans in range of 1000s

~~Although half of the categorical features have unique value count under 10, the other half has unique value counts in 1000s. It'll not be wise to apply OHE here as it'll create sparse vector having len in 1000s~~ The encoding method depends on the algorithm under consideration. For instance, LR works pretty well with high dimentional data. OHE could give a good score in this case. But the same might not be suitable for algorithms that are affected by curse of dimensionality like KNN. Also, it might not work well on tree based models.

Numerical range for class label

Since the number of numerical features are 190, we'll only look into range of few features.

We are only looking into features Var21 to Var25

In [ ]:

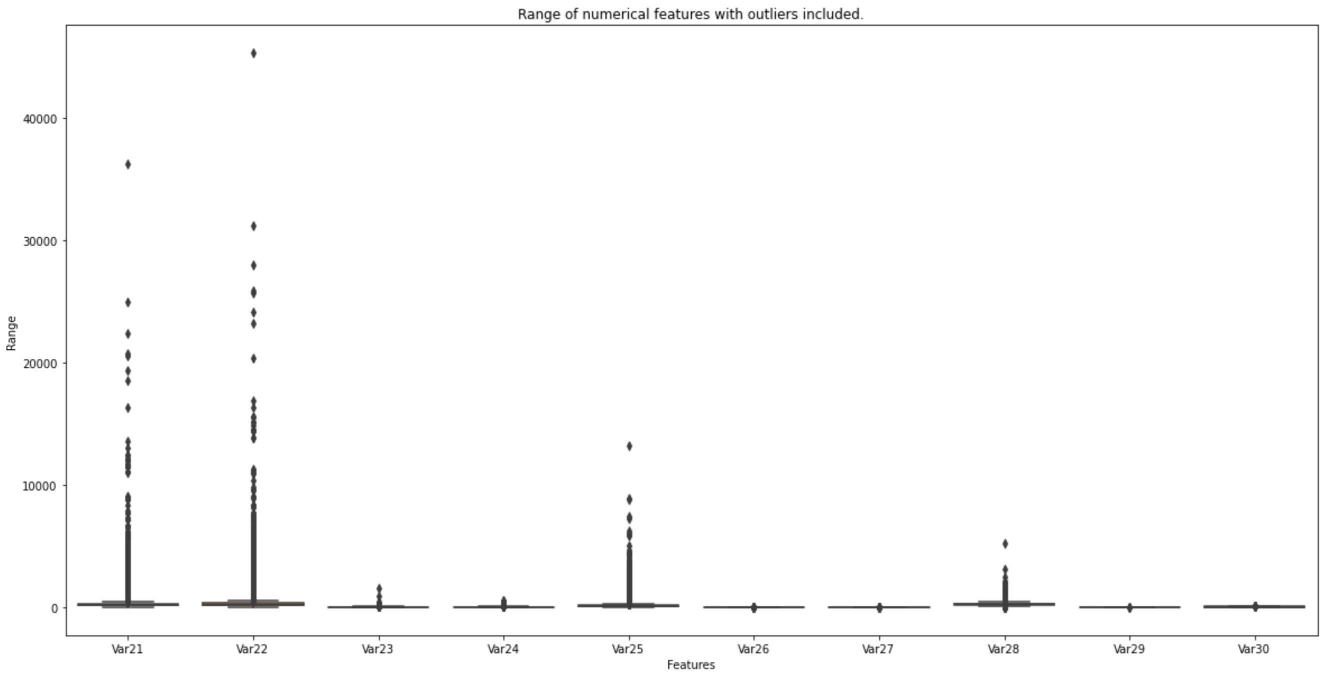
```
X_train_churn.iloc[:,20:30].head()
```

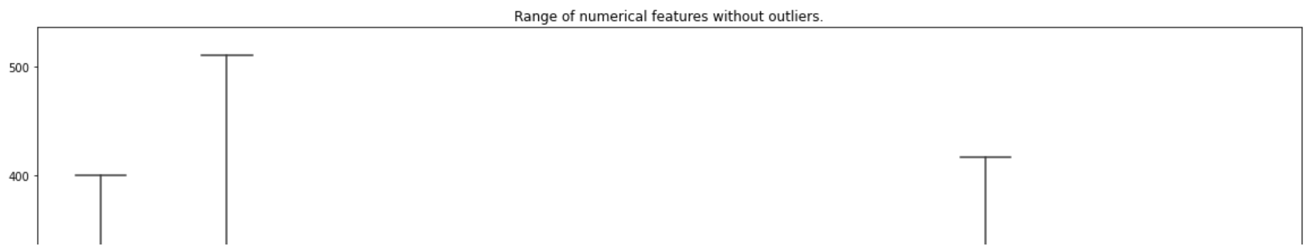
Out[ ]:

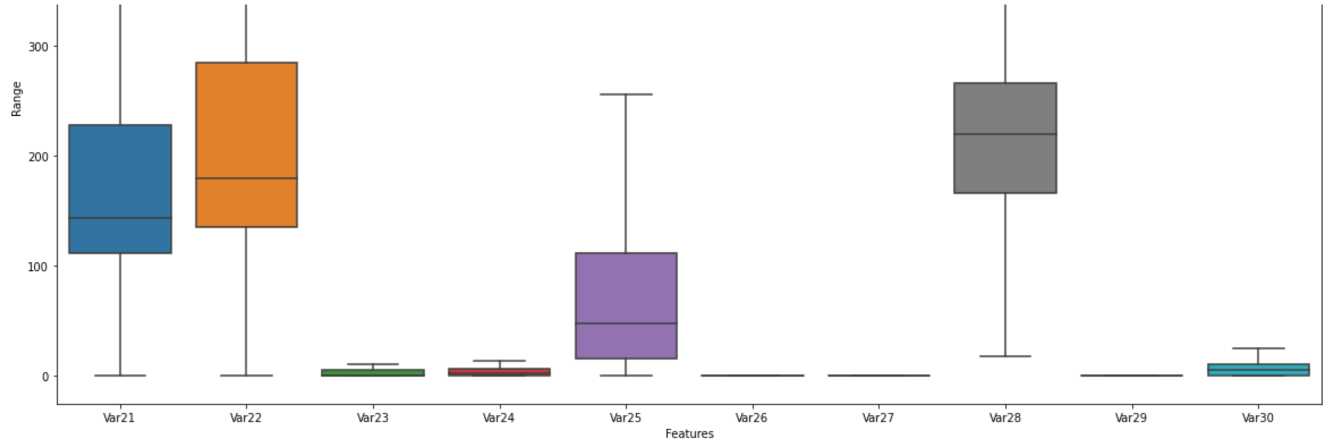
	Var21	Var22	Var23	Var24	Var25	Var26	Var27	Var28	Var29	Var30
13282	596.0	745.0	NaN	10.0	272.0	NaN	NaN	186.64	NaN	NaN
44013	140.0	175.0	NaN	0.0	8.0	NaN	NaN	321.60	NaN	NaN
30999	120.0	150.0	NaN	4.0	88.0	NaN	NaN	186.64	NaN	NaN
7521	536.0	670.0	NaN	20.0	192.0	NaN	NaN	213.36	NaN	NaN
17794	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	10.0

In [ ]:

```
# https://www.mikulskibartosz.name/how-to-remove-outliers-from-seaborn-boxplot-charts/
fig = plt.figure(figsize = (20, 10))
sns.boxplot(data = X_train_churn.iloc[:,20:30])
plt.title('Range of numerical features with outliers included.')
plt.xlabel('Features')
plt.ylabel('Range')
plt.show()
fig = plt.figure(figsize = (20, 10))
sns.boxplot(data = X_train_churn.iloc[:,20:30],showfliers = False)
plt.title('Range of numerical features without outliers.')
plt.xlabel('Features')
plt.ylabel('Range')
plt.show()
```







Observation:

- All features have different range.
- Var23 and Var24 have similar range.

Checking for pattern in missing values

In [ ]:

```
#https://towardsdatascience.com/missing-data-cfd9dbfd11b7
#https://towardsdatascience.com/all-about-missing-data-handling-b94b8b5d2184
#https://towardsdatascience.com/using-the-missingno-python-library-to-identify-and-visualise-missing-da
ta-prior-to-machine-learning-34c8c5b5f009
#https://github.com/ResidentMario/missingno
```

We'll check whether the NaNs value occur w.r.t a specific class or not

In [ ]:

```
data_with_labels = pd.concat([X_train_churn,y_train_churn],axis = 1)
```

In [ ]:

```
#https://stackoverflow.com/questions/53947196/groupby-class-and-count-missing-values-in-features
#https://stackoverflow.com/questions/39454542/divide-two-dataframes-with-python
# Percentage of NaNs w.r.t class
data_with_labels.isna().groupby(data_with_labels.Churn).sum().div(data_with_labels.Churn.value_counts()
,axis = 0) * 100
```

Out[ ]:

	Var1	Var2	Var3	Var4	Var5	Var6	Var7	Var8	Var9	Var10	V
Churn											
0	98.567266	97.382764	97.382764	96.686633	96.883600	11.545518	11.502347	100.0	98.567266	96.883600	97.38
1	98.876787	98.400272	98.400272	98.059905	98.127978	7.522124	7.828455	100.0	98.876787	98.127978	98.40

2 rows × 231 columns

Observation:

- NaN value doesn't occur for specific class.
- Percetage of NaN is uniform across classe.

Dropping columns with only NaNs value present

In [ ]:

```
temp_data = X_train_churn.drop(columns = all_nan_columns)
```

In [ ]:

```
#temp_data = temp_data.drop(columns = not_nan_columns)
```

In [ ]:

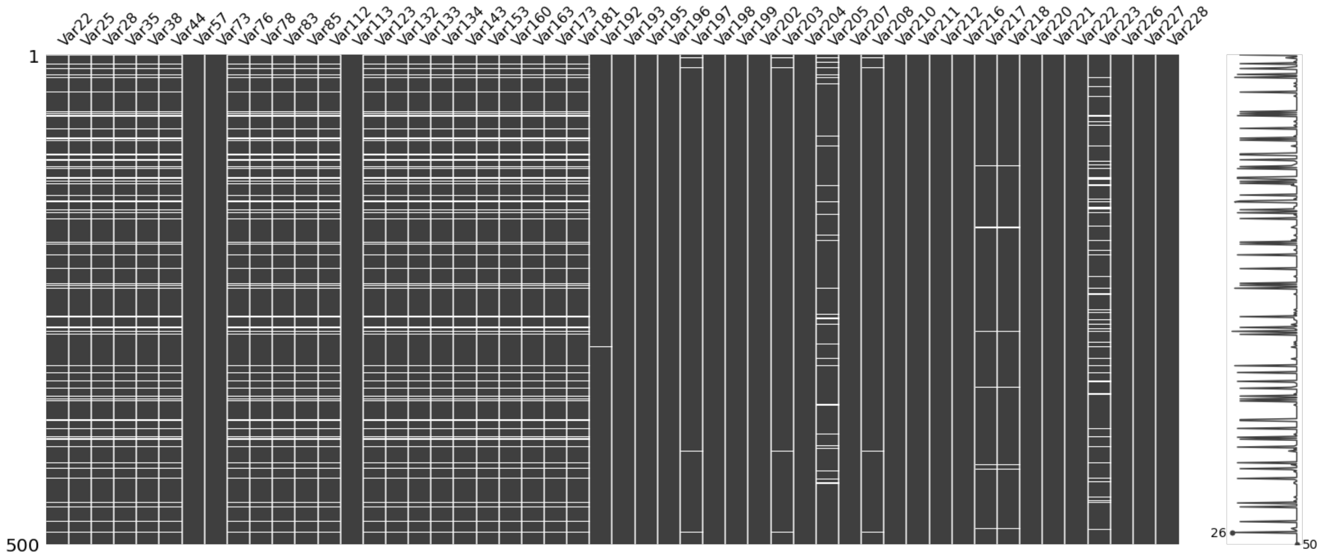
```
temp_data.shape
```

Out[ ]:

```
(40000, 212)
```

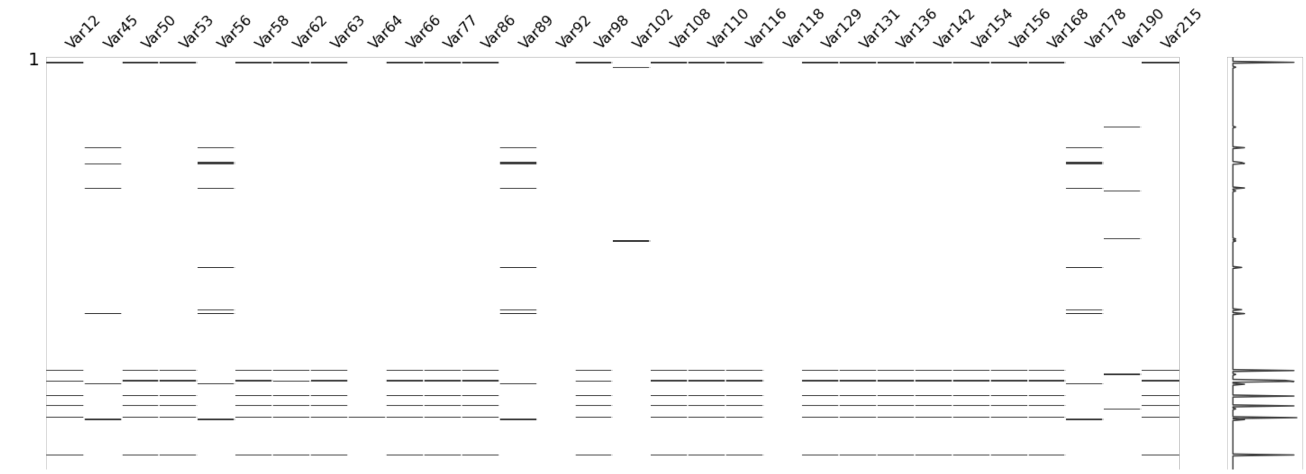
In [ ]:

```
filter_data = msno.nullity_filter(temp_data, filter='top',n = 50)
msno.matrix(filter_data.sample(500))
plt.show()
```



In [ ]:

```
filter_data = msno.nullity_filter(temp_data, filter='bottom',n = 30)
msno.matrix(filter_data.sample(500))
plt.show()
```



The white lines in above figure represent missing data.

Observation:

- You can see there is a patten of missingness of values.

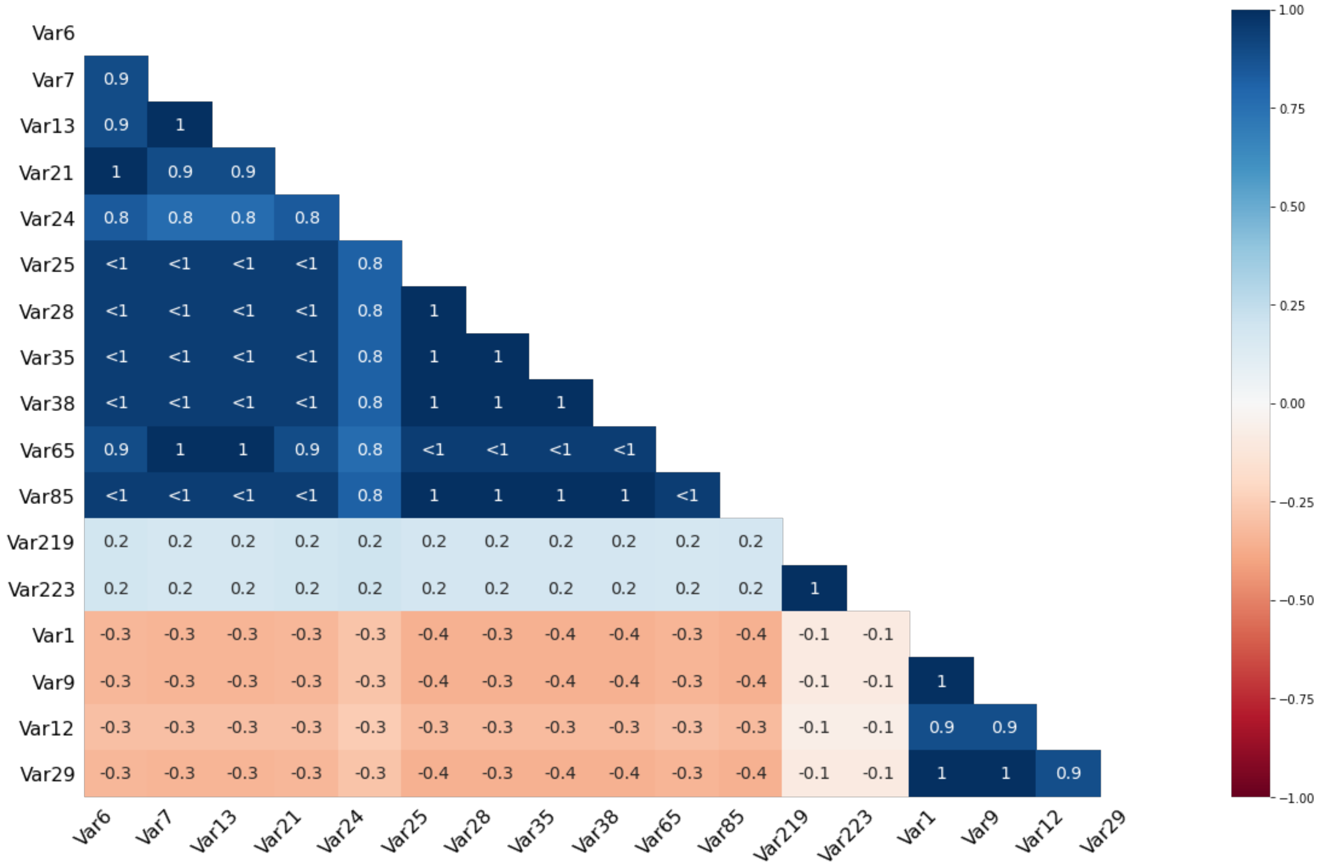
Checking for the correlation of missingness

In [ ]:

```
msno.heatmap(temp_data[['Var6','Var7','Var13','Var21','Var24','Var25','Var28','Var35','Var38','Var65','Var85','Var219','Var223','Var1','Var9','Var12','Var29']])
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2df207e450>
```



We took a handful of features to check whether there is a correlation in missingness of values.

- Nullity correlation ranges from -1 (if one variable appears the other definitely does not) to 0 (variables appearing or not appearing have no effect on one another) to 1 (if one variable appears the other definitely also does).

Observation:

- Most of the features we checked have value 1 meaning there is a correlation in missingness

Conclusion:

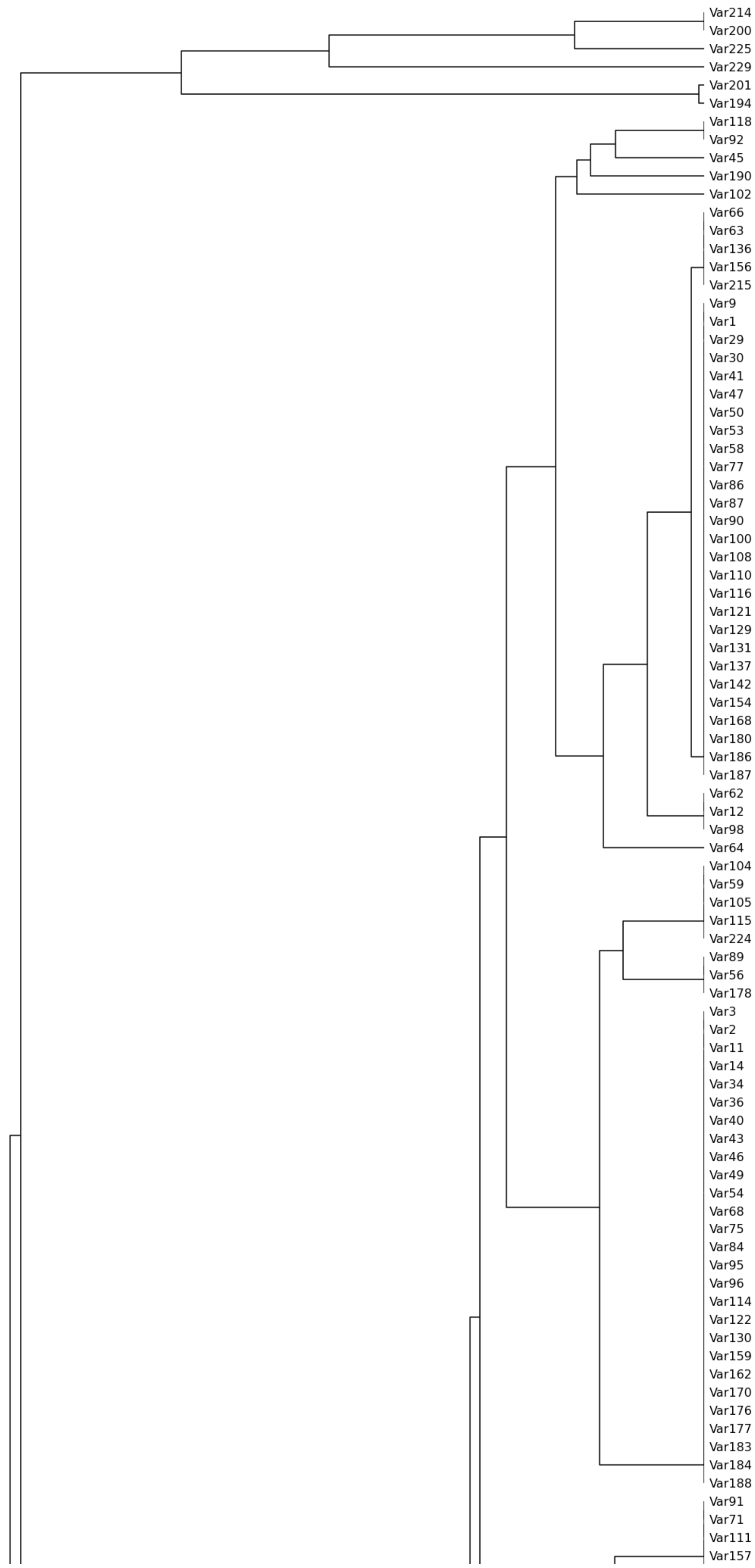
- As there is a correlation in missingness, we can rule out Missing Completely at random.

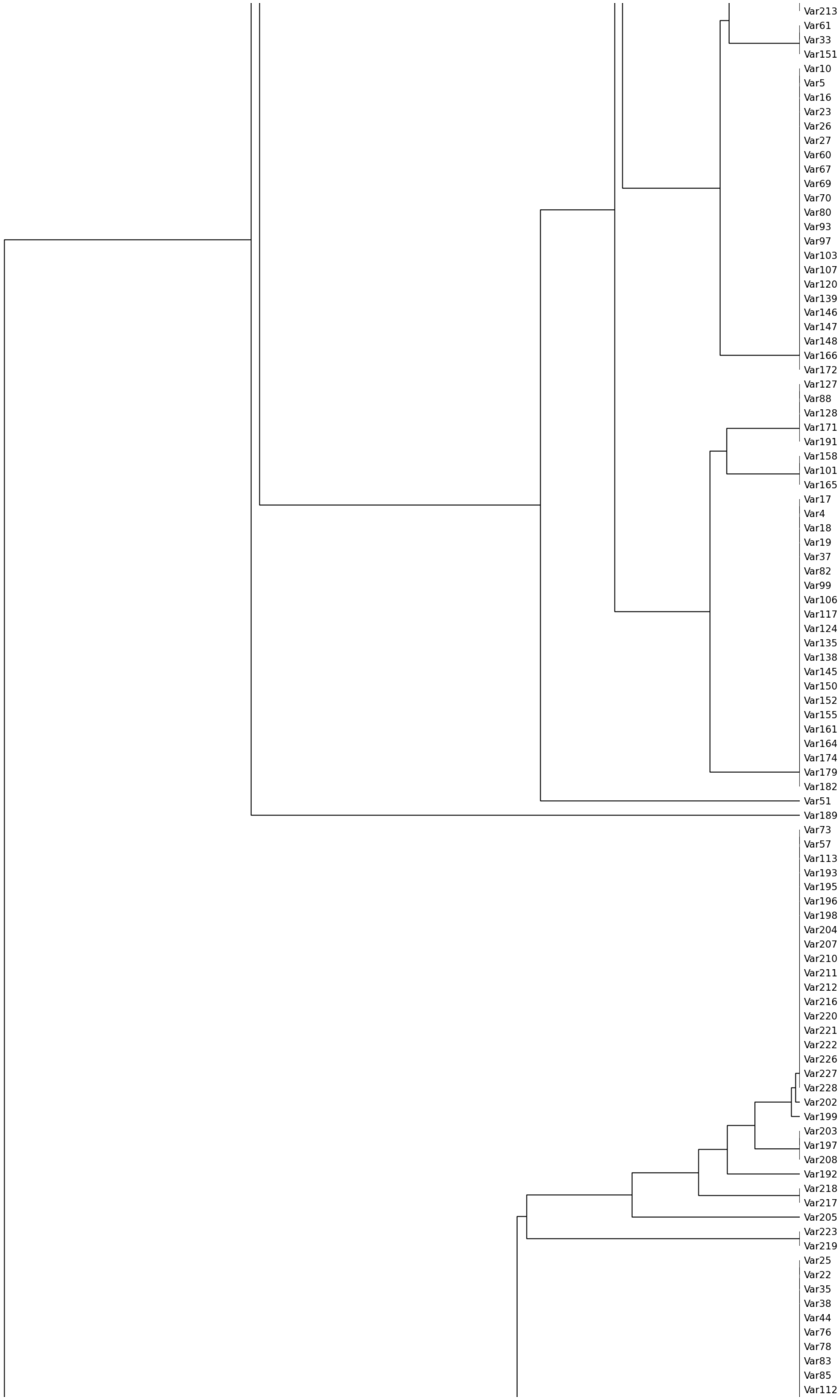
In [ ]:

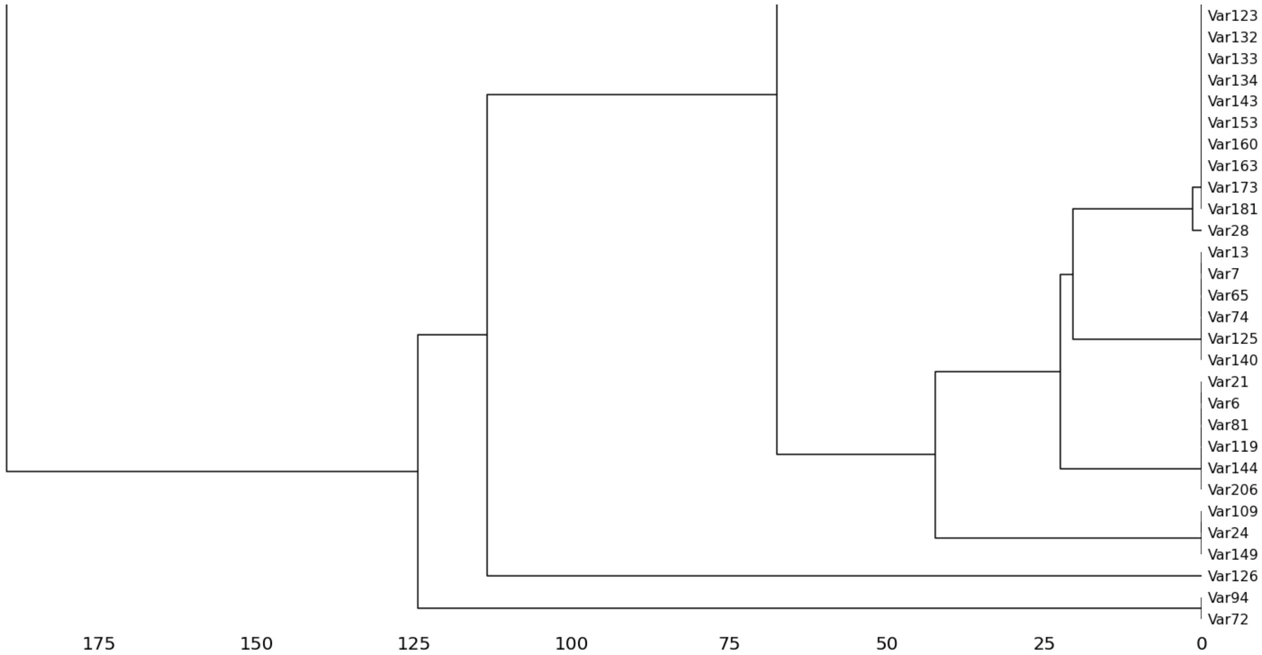
```
msno.dendrogram(temp_data)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2dfaba3d50>
```




Var213
Var61
Var33
Var151
Var10
Var5
Var16
Var23
Var26
Var27
Var60
Var67
Var69
Var70
Var80
Var93
Var97
Var103
Var107
Var120
Var139
Var146
Var147
Var148
Var166
Var172
Var127
Var88
Var128
Var171
Var191
Var158
Var101
Var165
Var17
Var4
Var18
Var19
Var37
Var82
Var99
Var106
Var117
Var124
Var135
Var138
Var145
Var150
Var152
Var155
Var161
Var164
Var174
Var179
Var182
Var51
Var189
Var73
Var57
Var113
Var193
Var195
Var196
Var198
Var204
Var207
Var210
Var211
Var212
Var216
Var220
Var221
Var222
Var226
Var227
Var228
Var202
Var199
Var203
Var197
Var208
Var192
Var218
Var217
Var205
Var223
Var219
Var25
Var22
Var35
Var38
Var44
Var76
Var78
Var83
Var85
Var112



Observation:

- Variable value which are linked together at 0 fully predict one another's presence i.e one variable might always be empty when another is filled, or they might always both be filled or both empty, and so on.
- There are lot of variable which are linked at 0 distance.

- You can see from the matrix above that in a feature group there is a pattern of missingness

<font color= "red"> Query: how to conclude whether it is MAR (Missing at random) or MNAR(Missing not at random)?

https://www.youtube.com/watch?v=YpqUbirqFxQ

https://www.youtube.com/watch?v=ACN29i_fqkk

https://www.youtube.com/watch?v=asyJCVLV4LI

To check if the missing data depends on the observed data (MAR), we'll put sub sample of missing data columns against sample of categorical columns with no missing data and see if data is missing for specific categorical value.

In [ ]:

```
missing_data_cols = ['Var123','Var132','Var133','Var143','Var153','Var160','Var163','Var173','Var181']
#cat_cols = ['Var192','Var193','Var195','Var196','Var197','Var198','Var199','Var202','Var203','Var204']
```

In [ ]:

```
cat_not_nan_cols = not_nan_columns[3:]
```

In [ ]:

```
num_not_nan = not_nan_columns[:3]
```

In [ ]:

```
all_cols = missing_data_cols + list(cat_not_nan_cols)
```

In [ ]:

```
missing_data = X_train_churn[all_cols]
```

In [ ]:

```
missing_data.head()
```

Out[ ]:

	Var123	Var132	Var133	Var143	Var153	Var160	Var163	Var173	Var181	Va
13282	60.0	0.0	365835.0	0.0	649576.0	144.0	221724.0	0.0	0.0	5QKljwyXr4MCZTEp7uAkS8PtI
44013	6.0	0.0	9245550.0	0.0	10672480.0	48.0	0.0	0.0	0.0	RO12
30999	0.0	0.0	4634955.0	0.0	10274360.0	22.0	2073600.0	0.0	0.0	2Knk1KF
7521	6.0	0.0	262870.0	0.0	480408.0	68.0	110844.0	0.0	0.0	RO12
17794	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	RO12

In [ ]:

```
missing_data.isna().any(axis = 1)
```

Out[ ]:

```
13282    False
44013    False
30999    False
7521     False
17794     True
          ...
34956    False
5456     False
37401    False
32888    False
317      False
Length: 40000, dtype: bool
```

In [ ]:

```
#https://stackoverflow.com/questions/14247586/how-to-select-rows-with-one-or-more-nulls-from-a-pandas-d
ataframe-without-listin
missing_data[missing_data.isna().any(axis = 1)][cat_not_nan_cols].nunique()
```

Out[ ]:

```
Var193       6
Var195       7
Var196       3
Var198    1127
Var204     100
Var207       6
Var210       5
Var211       2
Var212      16
Var216     155
Var220    1127
Var221       7
Var222    1127
Var226      23
Var227       7
Var228      10
dtype: int64
```

Checking for relation of missingness with numerical data

In [ ]:

```
num_not_nan
```

Out[ ]:

```
array(['Var57', 'Var73', 'Var113'], dtype=object)
```

In [ ]:

```
all_cols = missing_data_cols + list(num_not_nan)
```

In [ ]:

```
missing_data = X_train_churn[all_cols]
```

In [ ]:

```
missing_data.head()
```

Out[ ]:

	Var123	Var132	Var133	Var143	Var153	Var160	Var163	Var173	Var181	Var57	Var73	Var113
13282	60.0	0.0	365835.0	0.0	649576.0	144.0	221724.0	0.0	0.0	5.802820	214	-38241.44
44013	6.0	0.0	9245550.0	0.0	10672480.0	48.0	0.0	0.0	0.0	5.040376	28	210249.60
30999	0.0	0.0	4634955.0	0.0	10274360.0	22.0	2073600.0	0.0	0.0	5.946806	170	123881.20
7521	6.0	0.0	262870.0	0.0	480408.0	68.0	110844.0	0.0	0.0	0.411237	34	-1417852.00
17794	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.357830	4	-2951636.00

Dropping all nan

In [ ]:

```
non_missing_data = missing_data.dropna()
```

Checking min and max

In [ ]:

```
non_missing_data.min()
```

Out[ ]:

```
Var123     0.000000e+00
Var132     0.000000e+00
Var133     0.000000e+00
Var143     0.000000e+00
Var153     0.000000e+00
Var160     0.000000e+00
Var163     0.000000e+00
Var173     0.000000e+00
Var181     0.000000e+00
Var57      2.136296e-04
Var73      1.200000e+01
Var113    -9.803600e+06
dtype: float64
```

In [ ]:

```
non_missing_data.max()
```

Out[ ]:

```
Var123        13086.0
Var132          184.0
```

```
Var133      15009900.0
Var143            18.0
Var153      13757200.0
Var160          4862.0
Var163      14515200.0
Var173             6.0
Var181            49.0
Var57              7.0
Var73            264.0
Var113       9932480.0
dtype: float64
```

Only keeping nan data and then checking min and max of numerical var

In [ ]:

```
missing = missing_data[missing_data.isna().any(axis =1)].iloc[:,-3:]
```

In [ ]:

```
missing_data[missing_data.isna().any(axis =1)].min()
```

Out[ ]:

```
Var123               NaN
Var132               NaN
Var133               NaN
Var143               NaN
Var153               NaN
Var160               NaN
Var163               NaN
Var173               NaN
Var181               NaN
Var57       3.204443e-03
Var73       4.000000e+00
Var113     -9.684120e+06
dtype: float64
```

In [ ]:

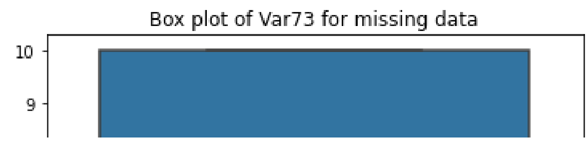
```
missing_data[missing_data.isna().any(axis =1)].max()
```

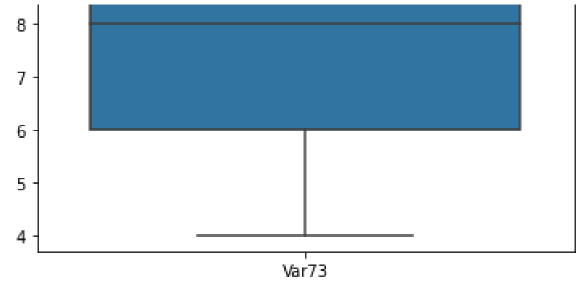
Out[ ]:

```
Var123            NaN
Var132            NaN
Var133            NaN
Var143            NaN
Var153            NaN
Var160            NaN
Var163            NaN
Var173            NaN
Var181            NaN
Var57             7.0
Var73            10.0
Var113      6239680.0
dtype: float64
```

In [ ]:

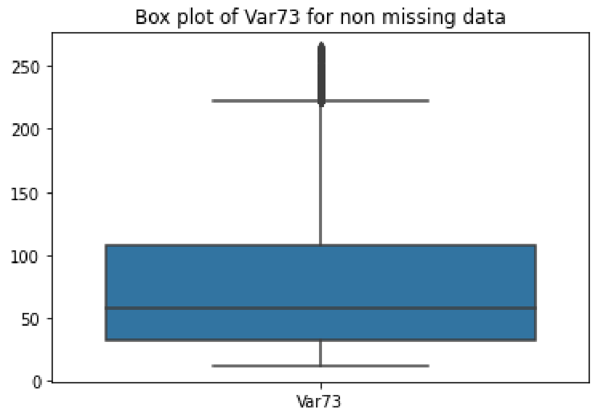
```
sns.boxplot(data = missing[['Var73']])
plt.title('Box plot of Var73 for missing data')
plt.show()
```





In [ ]:

```
sns.boxplot(data = non_missing_data[['Var73']])
plt.title('Box plot of Var73 for non missing data')
plt.show()
```



Observation:

- **For var73, if you look closely for non missing min and max, it is 12 and 264 resp. However max for missing data is 10.**

We can say that for the data missing the value of Var73 end at 10 but for data present value of Var73 starts at 12. This may be one of many other cases present in dataset.

Since there is pattern in missingness and a missingness depends on observed data and we can assume that this is Missing at Random (MAR).

Now that we have concluded that data is Missing at Random (MAR), we can either remove the NaN data or we can use imputation.

For removing data, we have:

- **Listwise deletion : Removes all data from an observation that has one or more missing values. Produces bias**
- **Pairwise deletion : Used in MCAR.**
- **Dropping variable : Dropping variables with having missing values %greater than 60%**

We'll be dropping variables followed by imputation.

Reference : https://towardsdatascience.com/how-to-handle-missing-data-8646b18db0d4

Handling NaNs

As we can see from the graph above, most of features have NaN values reaching close to 40k out of 40k datapoints. In order to handle that, we'll be removing features in which NaN value exceeds the threshold. We'll check for 50,60, 70, 80 percent for threshold value.

In [ ]:

```
nan_count_array = np.asarray(nan_count_array)
```

In [ ]:

```
print('Number of features which have NaN count less than 50 perc of original data: ',(nan_count_array <
.5*X_train_churn.shape[0]).sum())
```

```
Number of features which have NaN count less than 50 perc of original data:  69
```

In [ ]:

```
print('Number of features which have NaN count less than 60 perc of original data: ',(nan_count_array <
.6*X_train_churn.shape[0]).sum())
```

```
Number of features which have NaN count less than 60 perc of original data:  74
```

In [ ]:

```
print('Number of features which have NaN count less than 70 perc of original data: ',(nan_count_array <
.7*X_train_churn.shape[0]).sum())
```

```
Number of features which have NaN count less than 70 perc of original data:  74
```

In [ ]:

```
print('Number of features which have NaN count less than 80 perc of original data: ',(nan_count_array <
.8*X_train_churn.shape[0]).sum())
```

```
Number of features which have NaN count less than 80 perc of original data:  76
```

Observation:

- **When threshold is set at 50 perc, only 69 features have NaN count less than 50% of total data.**
- **For both 60 and 70 value of threshold, number of features remains same at 74.**
- **When threshold is set at 80%, number of features that satify the condition are 76. An increase of two feature from last observation.**

We'll continue with 60% threshold and remove features which have NaN count more than 60%

In [ ]:

```
features = np.argwhere(nan_count_array < .6*X_train_churn.shape[0])
```

In [ ]:

```
features = features.flatten()
```

In [ ]:

```
features
```

Out[ ]:

```
array([  5,   6,  12,  20,  21,  23,  24,  27,  34,  37,  43,  56,  64,
        71,  72,  73,  75,  77,  80,  82,  84,  93, 108, 111, 112, 118,
       122, 124, 125, 131, 132, 133, 139, 142, 143, 148, 152, 159, 162,
       172, 180, 188, 191, 192, 194, 195, 196, 197, 198, 199, 201, 202,
       203, 204, 205, 206, 207, 209, 210, 211, 213, 215, 216, 217, 218,
       219, 220, 221, 222, 224, 225, 226, 227, 228])
```

In [ ]:

```
data_new = X_train_churn.iloc[:,features]
```

```
data_new_test = X_test_churn.iloc[:, features]
```

In [ ]:

```
X_test_churn = X_test_churn.iloc[:, features]
```

In [ ]:

```
data_new.head()
```

Out[ ]:

	Var6	Var7	Var13	Var21	Var22	Var24	Var25	Var28	Var35	Var38	Var44	Var57	Var65	Var72	Var73	V
13282	3458.0	7.0	2528.0	596.0	745.0	10.0	272.0	186.64	0.0	25536.0	0.0	5.802820	18.0	3.0	214	7.
44013	616.0	0.0	0.0	140.0	175.0	0.0	8.0	321.60	0.0	199926.0	0.0	5.040376	9.0	3.0	28	0.
30999	777.0	14.0	428.0	120.0	150.0	4.0	88.0	186.64	0.0	0.0	0.0	5.946806	36.0	12.0	170	2
7521	3416.0	7.0	124.0	536.0	670.0	20.0	192.0	213.36	0.0	1062.0	0.0	0.411237	9.0	3.0	34	3
17794	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.357830	NaN	NaN	4	N

In [ ]:

```
data_new_test.head()
```

Out[ ]:

	Var6	Var7	Var13	Var21	Var22	Var24	Var25	Var28	Var35	Var38	Var44	Var57	Var65	Var72	Var73
24242	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.429609	NaN	NaN	6
9248	1554.0	14.0	468.0	160.0	200.0	10.0	48.0	186.64	0.0	4002636.0	0.0	4.268105	18.0	6.0	152
16049	455.0	0.0	0.0	132.0	165.0	0.0	104.0	153.20	10.0	2356602.0	18.0	6.255074	9.0	3.0	26
36719	812.0	7.0	3820.0	144.0	180.0	8.0	160.0	186.64	0.0	0.0	0.0	3.068148	9.0	3.0	152
48490	721.0	7.0	1996.0	24.0	30.0	0.0	0.0	253.52	0.0	3441234.0	0.0	3.869259	9.0	NaN	106

In [ ]:

```
#https://www.kaggle.com/questions-and-answers/181332
#http://shakedzy.xyz/dython/modules/nominal/#associations

# nominal.associations(data_new,figsize=(50,50), num_num_assoc= 'spearman',cmap = 'GnBu',mark_columns=T
rue);
```

Observation:

- **There are instances where a feature is highly correlated to other features. e.g : for Var21 has a correlation coef of 1 with Var22.**

<font color = 'red'> Query: Should we remove the highly correlated feature? i.e having corr > 0.8

This answer to this depends on factors like type of algorithm your are considering, interpretability of your results, etc.

Go through this thread once: https://datascience.stackexchange.com/questions/24452/in-supervised-learning-why-is-it-bad-to-have-correlated-features

Depending on the various experiment settings you create, treat the collinear features accordingly

We'll not be removing collinear features as having collinear features may or may not improve model performance but it will not degrade its performance. Also, they may be chance that new features based on these collinear features may add some new information to the model.

Feature Groups

Plotting means of the features

In [ ]:

```
data_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 40000 entries, 13282 to 317
Data columns (total 74 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Var6    35500 non-null  float64
 1   Var7    35507 non-null  float64
 2   Var13   35507 non-null  float64
 3   Var21   35500 non-null  float64
 4   Var22   35924 non-null  float64
 5   Var24   34152 non-null  float64
 6   Var25   35924 non-null  float64
 7   Var28   35922 non-null  float64
 8   Var35   35924 non-null  float64
 9   Var38   35924 non-null  float64
 10  Var44   35924 non-null  float64
 11  Var57   40000 non-null  float64
 12  Var65   35507 non-null  float64
 13  Var72   22069 non-null  float64
 14  Var73   40000 non-null  int64
 15  Var74   35507 non-null  float64
 16  Var76   35924 non-null  float64
 17  Var78   35924 non-null  float64
 18  Var81   35500 non-null  float64
 19  Var83   35924 non-null  float64
 20  Var85   35924 non-null  float64
 21  Var94   22069 non-null  float64
 22  Var109  34152 non-null  float64
 23  Var112  35924 non-null  float64
 24  Var113  40000 non-null  float64
 25  Var119  35500 non-null  float64
 26  Var123  35924 non-null  float64
 27  Var125  35507 non-null  float64
 28  Var126  28865 non-null  float64
 29  Var132  35924 non-null  float64
 30  Var133  35924 non-null  float64
 31  Var134  35924 non-null  float64
 32  Var140  35507 non-null  float64
 33  Var143  35924 non-null  float64
 34  Var144  35500 non-null  float64
 35  Var149  34152 non-null  float64
 36  Var153  35924 non-null  float64
 37  Var160  35924 non-null  float64
 38  Var163  35924 non-null  float64
 39  Var173  35924 non-null  float64
 40  Var181  35924 non-null  float64
 41  Var189  16828 non-null  float64
 42  Var192  39711 non-null  object
 43  Var193  40000 non-null  object
 44  Var195  40000 non-null  object
 45  Var196  40000 non-null  object
 46  Var197  39886 non-null  object
 47  Var198  40000 non-null  object
 48  Var199  39996 non-null  object
 49  Var200  19694 non-null  object
 50  Var202  39999 non-null  object
 51  Var203  39886 non-null  object
 52  Var204  40000 non-null  object
```

```
 53  Var205  38445 non-null  object
 54  Var206  35500 non-null  object
 55  Var207  40000 non-null  object
 56  Var208  39886 non-null  object
 57  Var210  40000 non-null  object
 58  Var211  40000 non-null  object
 59  Var212  40000 non-null  object
 60  Var214  19694 non-null  object
 61  Var216  40000 non-null  object
 62  Var217  39436 non-null  object
 63  Var218  39436 non-null  object
 64  Var219  35827 non-null  object
 65  Var220  40000 non-null  object
 66  Var221  40000 non-null  object
 67  Var222  40000 non-null  object
 68  Var223  35827 non-null  object
 69  Var225  19093 non-null  object
 70  Var226  40000 non-null  object
 71  Var227  40000 non-null  object
 72  Var228  40000 non-null  object
 73  Var229  17259 non-null  object
dtypes: float64(41), int64(1), object(32)
memory usage: 22.9+ MB
```

In [ ]:

```
numerical_data = data_new.iloc[:,0:42]
numerical_data_test = data_new_test.iloc[:,:42]
```

In [ ]:

```
numerical_data.head()
```

Out[ ]:

	Var6	Var7	Var13	Var21	Var22	Var24	Var25	Var28	Var35	Var38	Var44	Var57	Var65	Var72	Var73	V
13282	3458.0	7.0	2528.0	596.0	745.0	10.0	272.0	186.64	0.0	25536.0	0.0	5.802820	18.0	3.0	214	7.
44013	616.0	0.0	0.0	140.0	175.0	0.0	8.0	321.60	0.0	199926.0	0.0	5.040376	9.0	3.0	28	0.
30999	777.0	14.0	428.0	120.0	150.0	4.0	88.0	186.64	0.0	0.0	0.0	5.946806	36.0	12.0	170	2
7521	3416.0	7.0	124.0	536.0	670.0	20.0	192.0	213.36	0.0	1062.0	0.0	0.411237	9.0	3.0	34	3
17794	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.357830	NaN	NaN	4	N

In [ ]:

```
numerical_data_test.head()
```

Out[ ]:

	Var6	Var7	Var13	Var21	Var22	Var24	Var25	Var28	Var35	Var38	Var44	Var57	Var65	Var72	Var73
24242	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.429609	NaN	NaN	6
9248	1554.0	14.0	468.0	160.0	200.0	10.0	48.0	186.64	0.0	4002636.0	0.0	4.268105	18.0	6.0	152
16049	455.0	0.0	0.0	132.0	165.0	0.0	104.0	153.20	10.0	2356602.0	18.0	6.255074	9.0	3.0	26
36719	812.0	7.0	3820.0	144.0	180.0	8.0	160.0	186.64	0.0	0.0	0.0	3.068148	9.0	3.0	152
48490	721.0	7.0	1996.0	24.0	30.0	0.0	0.0	253.52	0.0	3441234.0	0.0	3.869259	9.0	NaN	106

In [ ]:

```
numerical_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 40000 entries, 13282 to 317
Data columns (total 42 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Var6     35500 non-null  float64
 1   Var7     35507 non-null  float64
 2   Var13    35507 non-null  float64
 3   Var21    35500 non-null  float64
 4   Var22    35924 non-null  float64
 5   Var24    34152 non-null  float64
 6   Var25    35924 non-null  float64
 7   Var28    35922 non-null  float64
 8   Var35    35924 non-null  float64
 9   Var38    35924 non-null  float64
 10  Var44    35924 non-null  float64
 11  Var57    40000 non-null  float64
 12  Var65    35507 non-null  float64
 13  Var72    22069 non-null  float64
 14  Var73    40000 non-null  int64  
 15  Var74    35507 non-null  float64
 16  Var76    35924 non-null  float64
 17  Var78    35924 non-null  float64
 18  Var81    35500 non-null  float64
 19  Var83    35924 non-null  float64
 20  Var85    35924 non-null  float64
 21  Var94    22069 non-null  float64
 22  Var109   34152 non-null  float64
 23  Var112   35924 non-null  float64
 24  Var113   40000 non-null  float64
 25  Var119   35500 non-null  float64
 26  Var123   35924 non-null  float64
 27  Var125   35507 non-null  float64
 28  Var126   28865 non-null  float64
 29  Var132   35924 non-null  float64
 30  Var133   35924 non-null  float64
 31  Var134   35924 non-null  float64
 32  Var140   35507 non-null  float64
 33  Var143   35924 non-null  float64
 34  Var144   35500 non-null  float64
 35  Var149   34152 non-null  float64
 36  Var153   35924 non-null  float64
 37  Var160   35924 non-null  float64
 38  Var163   35924 non-null  float64
 39  Var173   35924 non-null  float64
 40  Var181   35924 non-null  float64
 41  Var189   16828 non-null  float64
dtypes: float64(41), int64(1)
memory usage: 13.1 MB
```

In [ ]:

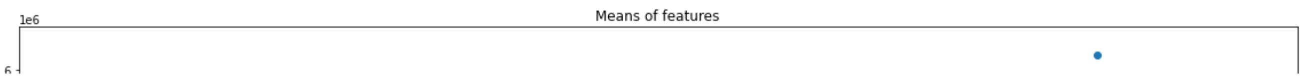
```
means = numerical_data.mean()
```

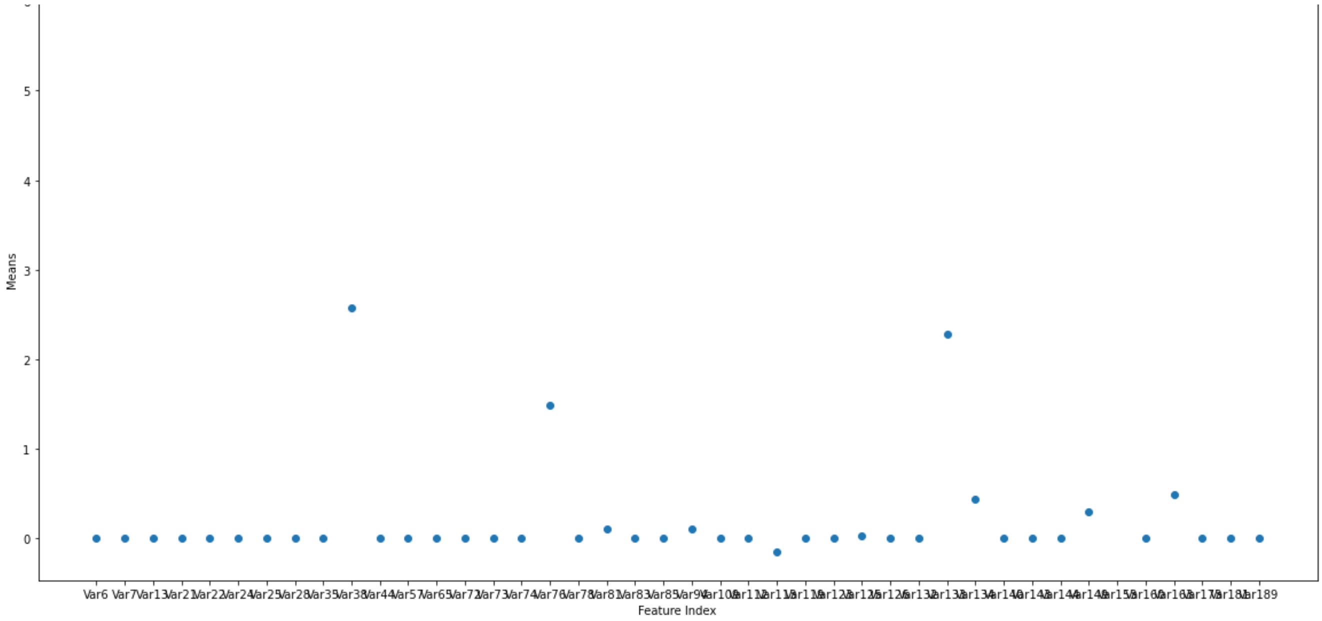
In [ ]:

```
means_test = numerical_data_test.mean()
```

In [ ]:

```
plt.figure(figsize = (20, 10))
plt.scatter(numerical_data.columns,means)
plt.title('Means of features')
plt.xlabel('Feature Index')
plt.ylabel('Means')
plt.show()
```





Observation:

- **Most of means on scale are close to 0.**
- **Only 4 features have mean > 1 million**

Let's try again by removing means > 1000000

In [ ]:

```
filter_means = means[means < 1000000]
```

In [ ]:

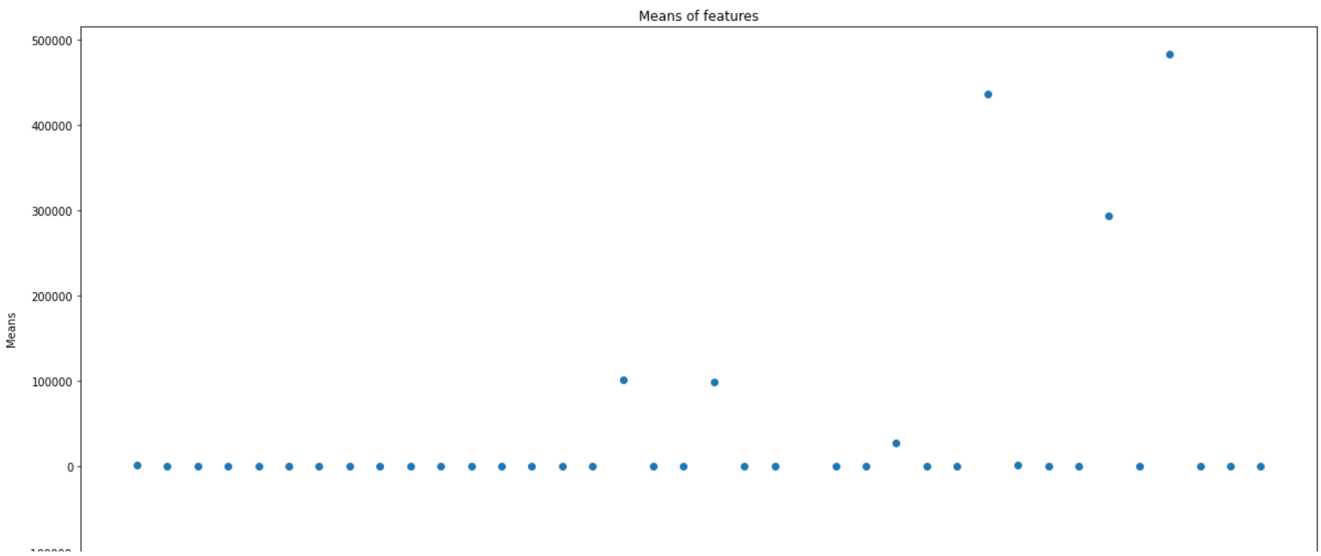
```
filter_means.shape
```

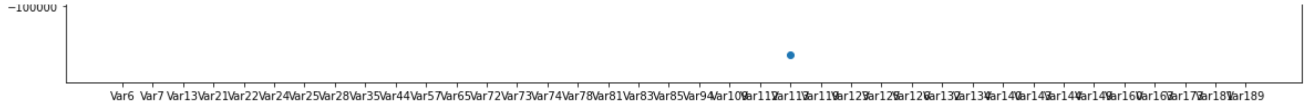
Out[ ]:

```
(38,)
```

In [ ]:

```
plt.figure(figsize = (20, 10))
plt.scatter(filter_means.index, filter_means)
plt.title('Means of features')
plt.ylabel('Means')
plt.show()
```





Observation:

- **Out of 42 numerical features, 38 are under mean of 1 million**
- **Most of the means are concentrated in region < 1 mil and close to 0**

Let's plot region under 10k

In [ ]:

```
filter_means = means[(means < 10000) & (means > 0)]
```

In [ ]:

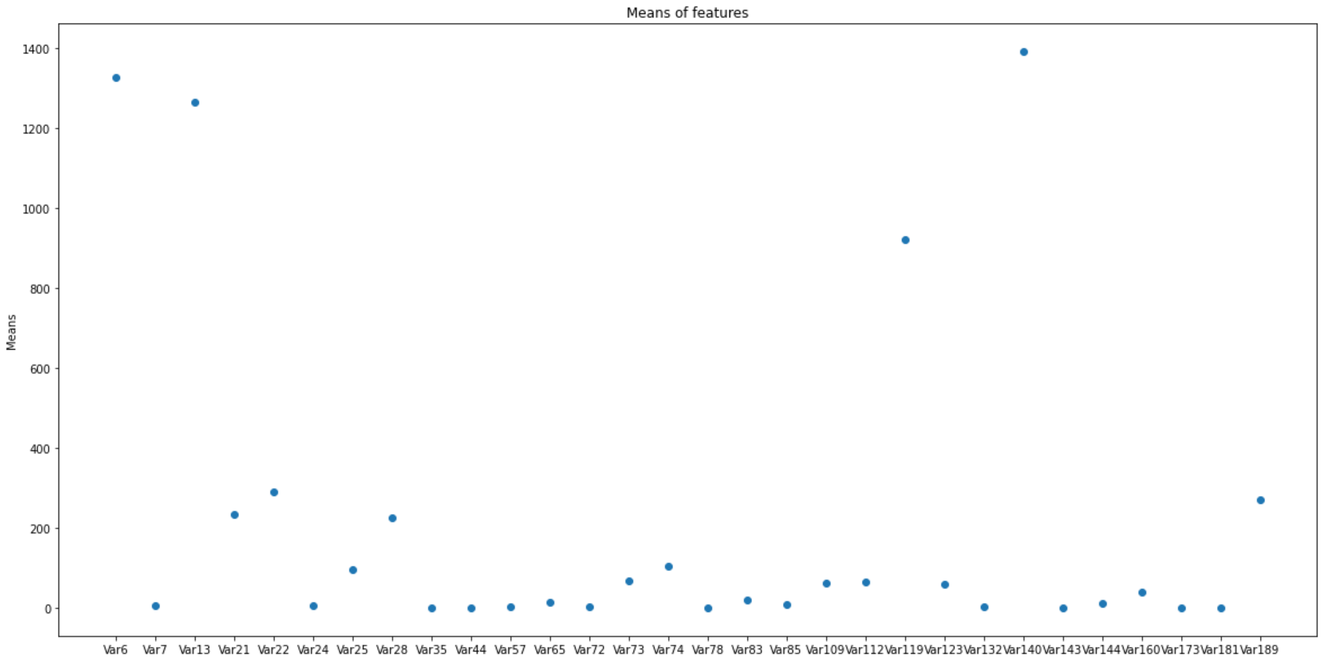
```
filter_means.shape
```

Out[ ]:

```
(30,)
```

In [ ]:

```
plt.figure(figsize = (20, 10))
plt.scatter(filter_means.index, filter_means)
plt.title('Means of features')
plt.ylabel('Means')
plt.show()
```



Observation:

- **There are 30 points which lie under 10k.**
- **Most of the points are concentrated under 400**

let's observation area under mean of 400

In [ ]:

```
filter_means = means[(means < 400) & (means > 0)]
```

In [ ]:

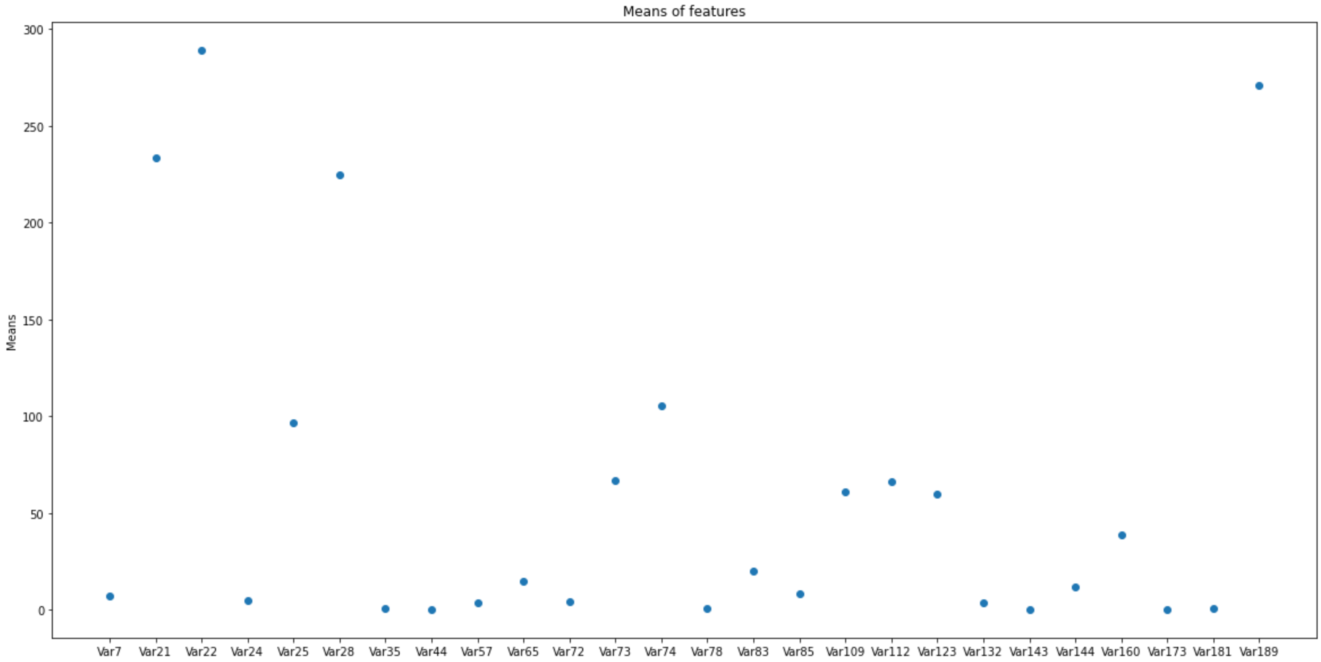
```
filter_means.shape
```

Out[ ]:

```
(26,)
```

In [ ]:

```
plt.figure(figsize = (20, 10))
plt.scatter(filter_means.index, filter_means)
plt.title('Means of features')
plt.ylabel('Means')
plt.show()
```



Observation:

- **Most of the means are concentrated under 50.**

In [ ]:

```
filter_means = means[(means < 50) & (means > 0)]
```

In [ ]:

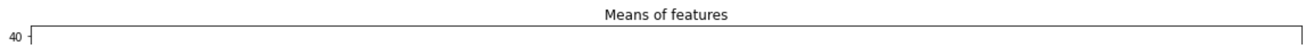
```
filter_means.shape
```

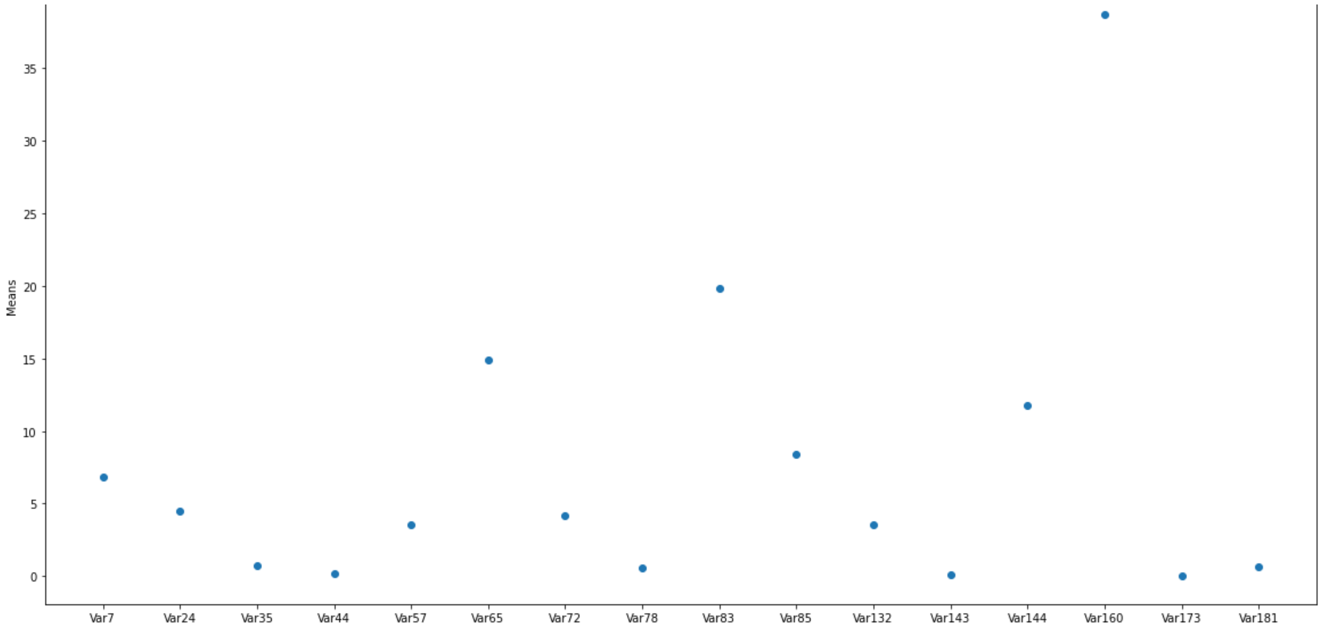
Out[ ]:

```
(16,)
```

In [ ]:

```
plt.figure(figsize = (20, 10))
plt.scatter(filter_means.index, filter_means)
plt.title('Means of features')
plt.ylabel('Means')
plt.show()
```





Observation:

- **Out of 42 numerical features, 26 have mean under 400.**
- **Out of 42 numerical features, 16 features have mean under 50.**
- **14 of the features have mean under 20.**

<font color= 'red'>Query: How does feature groups help us ?

```
Insight could help you create new features.
```

<font color = 'red'>Query: How does means help in identifying feature groups ?

```
We can form a feature group for features having similar means and use that feature group to gene
rate new features. for e.g: a a new feature which is average value of features having mean under
20.
```

We'll be making 2 new feature groups i.e

1. **Features having means under 200 and greater than 0**
2. **Features having means under 20 and greater than 0**

In [ ]:

```
means_test
```

Out[ ]:

```
Var6      1.328281e+03
Var7      6.781885e+00
Var13     1.192371e+03
Var21     2.380074e+02
Var22     2.946785e+02
Var24     4.530982e+00
Var25     9.736936e+01
Var28     2.231723e+02
Var35     7.119224e-01
Var38     2.609874e+06
Var44     1.697364e-01
Var57     3.511862e+00
Var65     1.474436e+01
Var72     4.163574e+00
Var73     6.612300e+01
Var74     9.667489e+01
Var76     1.497728e+06
Var78     5.283997e-01
```

```
Var81     1.057588e+05
Var83     2.062865e+01
Var85     8.636594e+00
Var94     9.823136e+04
Var109    6.126804e+01
Var112    6.661961e+01
Var113   -1.349349e+05
Var119    8.954002e+02
Var123    6.254108e+01
Var125    2.940204e+04
Var126   -6.170478e-01
Var132    3.517812e+00
Var133    2.261058e+06
Var134    4.399443e+05
Var140    1.344609e+03
Var143    6.882100e-02
Var144    1.147899e+01
Var149    3.006562e+05
Var153    6.217543e+06
Var160    3.913908e+01
Var163    4.956880e+05
Var173    7.940885e-03
Var181    6.083600e-01
Var189    2.681173e+02
dtype: float64
```

In [ ]:

```
feature_group_200 = means[(means < 200) & (means > 0)]
```

In [ ]:

```
feature_group_200 = list(feature_group_200.index)
```

In [ ]:

```
feature_group_50 = means[(means < 50) & (means > 0)]
```

In [ ]:

```
feature_group_50 = list(feature_group_50.index)
```

In [ ]:

```
with open('feature_group_200.pickle', 'wb') as handle:
    pickle.dump(feature_group_200, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [ ]:

```
with open('feature_group_50.pickle', 'wb') as handle:
    pickle.dump(feature_group_50, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [ ]:

```
feature_group_50
```

Out[ ]:

```
['Var7',
 'Var24',
 'Var35',
 'Var44',
 'Var57',
 'Var65',
 'Var72',
 'Var78',
 'Var83',
```

```
 'Var85',
 'Var132',
 'Var143',
 'Var144',
 'Var160',
 'Var173',
 'Var181']
```

Clustering of features

In [ ]:

```python
#https://medium.com/analytics-vidhya/gowers-distance-899f9c4bd553
#https://towardsdatascience.com/clustering-datasets-having-both-numerical-and-categorical-variables-ed91cdca0677
```

In [ ]:

```python
data_new.head()
```

Out[ ]:

	Var6	Var7	Var13	Var21	Var22	Var24	Var25	Var28	Var35	Var38	Var44	Var57	Var65	Var72	Var73	V
13282	3458.0	7.0	2528.0	596.0	745.0	10.0	272.0	186.64	0.0	25536.0	0.0	5.802820	18.0	3.0	214	7.
44013	616.0	0.0	0.0	140.0	175.0	0.0	8.0	321.60	0.0	199926.0	0.0	5.040376	9.0	3.0	28	0.
30999	777.0	14.0	428.0	120.0	150.0	4.0	88.0	186.64	0.0	0.0	0.0	5.946806	36.0	12.0	170	2
7521	3416.0	7.0	124.0	536.0	670.0	20.0	192.0	213.36	0.0	1062.0	0.0	0.411237	9.0	3.0	34	3
17794	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.357830	NaN	NaN	4	N

In [ ]:

```python
data_new_test.head()
```

Out[ ]:

	Var6	Var7	Var13	Var21	Var22	Var24	Var25	Var28	Var35	Var38	Var44	Var57	Var65	Var72	Var73
24242	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.429609	NaN	NaN	6
9248	1554.0	14.0	468.0	160.0	200.0	10.0	48.0	186.64	0.0	4002636.0	0.0	4.268105	18.0	6.0	152
16049	455.0	0.0	0.0	132.0	165.0	0.0	104.0	153.20	10.0	2356602.0	18.0	6.255074	9.0	3.0	26
36719	812.0	7.0	3820.0	144.0	180.0	8.0	160.0	186.64	0.0	0.0	0.0	3.068148	9.0	3.0	152
48490	721.0	7.0	1996.0	24.0	30.0	0.0	0.0	253.52	0.0	3441234.0	0.0	3.869259	9.0	NaN	106

Before we start off with clustering, we need to deal with NaN data. For numerical data, we'll perform mean imputation and for categorical data, we'll consider NaN as separate category.

In [ ]:

```python
data_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 40000 entries, 13282 to 317
Data columns (total 74 columns):
 #   Column  Non-Null Count  Dtype
```

```
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Var6    35500 non-null  float64
 1   Var7    35507 non-null  float64
 2   Var13   35507 non-null  float64
 3   Var21   35500 non-null  float64
 4   Var22   35924 non-null  float64
 5   Var24   34152 non-null  float64
 6   Var25   35924 non-null  float64
 7   Var28   35922 non-null  float64
 8   Var35   35924 non-null  float64
 9   Var38   35924 non-null  float64
 10  Var44   35924 non-null  float64
 11  Var57   40000 non-null  float64
 12  Var65   35507 non-null  float64
 13  Var72   22069 non-null  float64
 14  Var73   40000 non-null  int64
 15  Var74   35507 non-null  float64
 16  Var76   35924 non-null  float64
 17  Var78   35924 non-null  float64
 18  Var81   35500 non-null  float64
 19  Var83   35924 non-null  float64
 20  Var85   35924 non-null  float64
 21  Var94   22069 non-null  float64
 22  Var109  34152 non-null  float64
 23  Var112  35924 non-null  float64
 24  Var113  40000 non-null  float64
 25  Var119  35500 non-null  float64
 26  Var123  35924 non-null  float64
 27  Var125  35507 non-null  float64
 28  Var126  28865 non-null  float64
 29  Var132  35924 non-null  float64
 30  Var133  35924 non-null  float64
 31  Var134  35924 non-null  float64
 32  Var140  35507 non-null  float64
 33  Var143  35924 non-null  float64
 34  Var144  35500 non-null  float64
 35  Var149  34152 non-null  float64
 36  Var153  35924 non-null  float64
 37  Var160  35924 non-null  float64
 38  Var163  35924 non-null  float64
 39  Var173  35924 non-null  float64
 40  Var181  35924 non-null  float64
 41  Var189  16828 non-null  float64
 42  Var192  39711 non-null  object
 43  Var193  40000 non-null  object
 44  Var195  40000 non-null  object
 45  Var196  40000 non-null  object
 46  Var197  39886 non-null  object
 47  Var198  40000 non-null  object
 48  Var199  39996 non-null  object
 49  Var200  19694 non-null  object
 50  Var202  39999 non-null  object
 51  Var203  39886 non-null  object
 52  Var204  40000 non-null  object
 53  Var205  38445 non-null  object
 54  Var206  35500 non-null  object
 55  Var207  40000 non-null  object
 56  Var208  39886 non-null  object
 57  Var210  40000 non-null  object
 58  Var211  40000 non-null  object
 59  Var212  40000 non-null  object
 60  Var214  19694 non-null  object
 61  Var216  40000 non-null  object
 62  Var217  39436 non-null  object
 63  Var218  39436 non-null  object
 64  Var219  35827 non-null  object
 65  Var220  40000 non-null  object
 66  Var221  40000 non-null  object
 67  Var222  40000 non-null  object
 68  Var223  35827 non-null  object
 69  Var225  19093 non-null  object
 70  Var226  40000 non-null  object
 71  Var227  40000 non-null  object
 72  Var228  40000 non-null  object
 73  Var229  17259 non-null  object
dtypes: float64(41), int64(1), object(32)
```

```
memory usage: 22.9+ MB
```

In [ ]:

```
data_new.mean()
```

Out[ ]:

```
Var6      1.325971e+03
Var7      6.816459e+00
Var13     1.264142e+03
Var21     2.336365e+02
Var22     2.891265e+02
Var24     4.502108e+00
Var25     9.669012e+01
Var28     2.248447e+02
Var35     7.180436e-01
Var38     2.571342e+06
Var44     1.661007e-01
Var57     3.512424e+00
Var65     1.490030e+01
Var72     4.197472e+00
Var73     6.677060e+01
Var74     1.054191e+02
Var76     1.488242e+06
Var78     5.362989e-01
Var81     1.024081e+05
Var83     1.987084e+01
Var85     8.416713e+00
Var94     9.878167e+04
Var109    6.079293e+01
Var112    6.612048e+01
Var113   -1.578645e+05
Var119    9.213462e+02
Var123    5.959414e+01
Var125    2.750573e+04
Var126   -5.380911e-01
Var132    3.526333e+00
Var133    2.276730e+06
Var134    4.366832e+05
Var140    1.390502e+03
Var143    5.528338e-02
Var144    1.179051e+01
Var149    2.934735e+05
Var153    6.172988e+06
Var160    3.871818e+01
Var163    4.836524e+05
Var173    6.569424e-03
Var181    6.122369e-01
Var189    2.706468e+02
dtype: float64
```

In [ ]:

```
data_impute = data_new.iloc[:,0:42].fillna(data_new.mean())
```

In [ ]:

```
data_impute_test = data_new_test.iloc[:,0:42].fillna(data_new_test.mean())
```

In [ ]:

```
data_new_imputed = pd.concat([data_impute, data_new.iloc[:,42:].fillna('Others')], axis =1)
```

In [ ]:

```
data_new_imputed_test = pd.concat([data_impute_test, data_new_test.iloc[:,42:].fillna('Others')], axis
=1)
```

In [ ]:

```
data_new_imputed.head()
```

Out[ ]:

	Var6	Var7	Var13	Var21	Var22	Var24	Var25	Var28	Var35	
13282	3458.000000	7.000000	2528.000000	596.000000	745.000000	10.000000	272.000000	186.640000	0.000000	2.55
44013	616.000000	0.000000	0.000000	140.000000	175.000000	0.000000	8.000000	321.600000	0.000000	1.99
30999	777.000000	14.000000	428.000000	120.000000	150.000000	4.000000	88.000000	186.640000	0.000000	0.00
7521	3416.000000	7.000000	124.000000	536.000000	670.000000	20.000000	192.000000	213.360000	0.000000	1.06
17794	1325.971155	6.816459	1264.142394	233.636507	289.126489	4.502108	96.690124	224.844725	0.718044	2.57

In [ ]:

```
data_new_imputed_test.head()
```

Out[ ]:

	Var6	Var7	Var13	Var21	Var22	Var24	Var25	Var28	Var35	
24242	1328.281017	6.781885	1192.371231	238.007357	294.678504	4.530982	97.369361	223.17231	0.711922	2.60
9248	1554.000000	14.000000	468.000000	160.000000	200.000000	10.000000	48.000000	186.64000	0.000000	4.00
16049	455.000000	0.000000	0.000000	132.000000	165.000000	0.000000	104.000000	153.20000	10.000000	2.35
36719	812.000000	7.000000	3820.000000	144.000000	180.000000	8.000000	160.000000	186.64000	0.000000	0.00
48490	721.000000	7.000000	1996.000000	24.000000	30.000000	0.000000	0.000000	253.52000	0.000000	3.44

Since our data contain both categorical and numerical features, we'll first convert our Categorical Data to numerical using ordinal encoding.

In [ ]:

```
encoder = OrdinalEncoder(handle_unknown = 'use_encoded_value', unknown_value = -1)
```

In [ ]:

```
data_new_imputed.iloc[:, 42:].head()
```

Out[ ]:

	Var192	Var193	Var195	Var196	Var197	Var198	Var199	Var200	Var2
13282	oUPBcmzkzH	5QKljwyXr4MCZTEp7uAkS8PtBLcn	taul	1K8T	IK27	fhk21Ss	r83_sZi	Xuaegi4	kk_f
44013	52Iq9ayE15	RO12	taul	1K8T	z32I	UsSOoyT	nQUq7hGe64	Others	jrUy
30999	a4vPe2fHUn	2Knk1KF	taul	1K8T	TyGI	THRJJYr	r83_sZi	Sc4mZtf	sOY
7521	EsYq9aX0Db	RO12	taul	1K8T	L80O	8K14q6X	PaagavI	Others	9n6
17794	1YVvyx7IEC	RO12	taul	1K8T	FgS1	oKsWccX	Gai9IEF2Fr	Others	Ib2I

In [ ]:

```
encoder.fit(data_new_imputed.iloc[:,42:])
```

Out[ ]:

```
OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
```

In [ ]:

```
ordinal_features = encoder.transform(data_new_imputed.iloc[:,42:])
```

In [ ]:

```
ordinal_features_test = encoder.transform(data_new_imputed_test.iloc[:,42:])
```

In [ ]:

```
ordinal_features.shape
```

Out[ ]:

```
(40000, 32)
```

In [ ]:

```
ordinal_features_test.shape
```

Out[ ]:

```
(10000, 32)
```

In [ ]:

```
numerical_features = data_new_imputed.iloc[:,0:42].values
numerical_features_test = data_new_imputed_test.iloc[:,0:42].values
```

In [ ]:

```
numerical_features.shape
```

Out[ ]:

```
(40000, 42)
```

In [ ]:

```
numerical_features_test.shape
```

Out[ ]:

```
(10000, 42)
```

In [ ]:

```
final_features = np.hstack((numerical_features, ordinal_features))
```

In [ ]:

```
final_features.shape
```

Out[ ]:

```
(40000, 74)
```

In [ ]:

```
final_features_test = np.hstack((numerical_features_test, ordinal_features_test))
```

In [ ]:

```
final_features_test.shape
```

Out[ ]:

```
(10000, 74)
```

Clustering of points

Reference: https://towardsdatascience.com/how-to-create-new-features-using-clustering-4ae772387290

In [ ]:

```
train_labels = []
test_labels = []
for i in range(2,7):
    kmeans = KMeans(n_clusters=i, n_jobs = -1)
    kmeans.fit(final_features)
    train_labels.append(kmeans.labels_)
    test_labels.append(kmeans.predict(final_features_test))
```

In [ ]:

```
# embedded_features = TSNE(n_jobs = -1).fit_transform(final_features)
```

In [ ]:

```
# for i in range(5):
#     plt.figure(figsize = (20,20))
#     plt.scatter(embedded_features[:,0], embedded_features[:,1], c= labels[i])
#     plt.title('Clustering of Features. Number of cluster: {}'.format(i+2))
#     plt.show()
```

Observation:

- **The above plot shows the datapoints divided in 2,3,4,5 and 6 cluster.**

We will use this cluster label as new feature.

<font color = 'red'> Query: How does clustering help in feature group?

you can assign cluster numbers to similar features (groups) to create a new feature. Some more areas can also be explored.

Finding Duplicate features

In [ ]:

```
#https://towardsdatascience.com/the-fastml-guide-9ada1bb761cf
duplicate_features = get_duplicate_features(data_new)
```

In [ ]:

```
duplicate_features.head()
```

Out[ ]:

	Desc	feature1	feature2
0	Duplicate Index	Var198	Var220
1	Duplicate Index	Var198	Var222
2	Duplicate Index	Var220	Var222

From the Description, we can see that although the values of two features are different but they occur at same index. Let's print them and see.

In [ ]:

```
data_new[data_new.Var198 == 'NldASpP'][['Var198','Var220','Var222']]
```

Out[ ]:

	Var198	Var220	Var222
49432	NldASpP	JFM1BiF	NKv4yOc
16700	NldASpP	JFM1BiF	NKv4yOc
40811	NldASpP	JFM1BiF	NKv4yOc
40771	NldASpP	JFM1BiF	NKv4yOc
17269	NldASpP	JFM1BiF	NKv4yOc
26141	NldASpP	JFM1BiF	NKv4yOc
36377	NldASpP	JFM1BiF	NKv4yOc
21746	NldASpP	JFM1BiF	NKv4yOc
17160	NldASpP	JFM1BiF	NKv4yOc
18954	NldASpP	JFM1BiF	NKv4yOc
690	NldASpP	JFM1BiF	NKv4yOc
6355	NldASpP	JFM1BiF	NKv4yOc
48929	NldASpP	JFM1BiF	NKv4yOc
2	NldASpP	JFM1BiF	NKv4yOc
9423	NldASpP	JFM1BiF	NKv4yOc
43096	NldASpP	JFM1BiF	NKv4yOc
36477	NldASpP	JFM1BiF	NKv4yOc
47382	NldASpP	JFM1BiF	NKv4yOc
24119	NldASpP	JFM1BiF	NKv4yOc
20568	NldASpP	JFM1BiF	NKv4yOc
4878	NldASpP	JFM1BiF	NKv4yOc
21499	NldASpP	JFM1BiF	NKv4yOc
3800	NldASpP	JFM1BiF	NKv4yOc
23993	NldASpP	JFM1BiF	NKv4yOc
16980	NldASpP	JFM1BiF	NKv4yOc
15642	NldASpP	JFM1BiF	NKv4yOc
41830	NldASpP	JFM1BiF	NKv4yOc
21672	NldASpP	JFM1BiF	NKv4yOc
2322	NldASpP	JFM1BiF	NKv4yOc
16570	NldASpP	JFM1BiF	NKv4yOc
46360	NldASpP	JFM1BiF	NKv4yOc
38508	NldASpP	JFM1BiF	NKv4yOc

	Var198	Var220	Var222
156	NIdASpP	JFM1BiF	NKv4yOc
4418	NIdASpP	JFM1BiF	NKv4yOc
30843	NIdASpP	JFM1BiF	NKv4yOc
39116	NIdASpP	JFM1BiF	NKv4yOc
27003	NIdASpP	JFM1BiF	NKv4yOc
15538	NIdASpP	JFM1BiF	NKv4yOc
40437	NIdASpP	JFM1BiF	NKv4yOc
8559	NIdASpP	JFM1BiF	NKv4yOc
29914	NIdASpP	JFM1BiF	NKv4yOc
39608	NIdASpP	JFM1BiF	NKv4yOc
46569	NIdASpP	JFM1BiF	NKv4yOc
33827	NIdASpP	JFM1BiF	NKv4yOc
40080	NIdASpP	JFM1BiF	NKv4yOc
43960	NIdASpP	JFM1BiF	NKv4yOc
18737	NIdASpP	JFM1BiF	NKv4yOc
31782	NIdASpP	JFM1BiF	NKv4yOc
43492	NIdASpP	JFM1BiF	NKv4yOc
34860	NIdASpP	JFM1BiF	NKv4yOc
7282	NIdASpP	JFM1BiF	NKv4yOc
40704	NIdASpP	JFM1BiF	NKv4yOc
37386	NIdASpP	JFM1BiF	NKv4yOc
36419	NIdASpP	JFM1BiF	NKv4yOc

In [ ]:

```
data_new[data_new.Var198 == 'ka_ns41'][['Var198','Var220','Var222']]
```

Out[ ]:

	Var198	Var220	Var222
34887	ka_ns41	1YVfGrO	fXVEsaq
20728	ka_ns41	1YVfGrO	fXVEsaq
45121	ka_ns41	1YVfGrO	fXVEsaq
36299	ka_ns41	1YVfGrO	fXVEsaq
18081	ka_ns41	1YVfGrO	fXVEsaq
...	...	...	...
20152	ka_ns41	1YVfGrO	fXVEsaq
32893	ka_ns41	1YVfGrO	fXVEsaq
45894	ka_ns41	1YVfGrO	fXVEsaq
46387	ka_ns41	1YVfGrO	fXVEsaq
230	ka_ns41	1YVfGrO	fXVEsaq

100 rows × 3 columns

Observation:

- **Although we didn't find any duplicate features but there are 3 features for which value are different but they have same mapping.**

- For ex: For column Var198, value 'ka_ns41' always occur with '1YVfGrO' (Var220) and 'fXVEsaq' (Var222)

<font color = 'red'> Query: Do we remove features with values having same mapping. If so, why?

The duplicate columns could be dropped Because they are the same things

Dropping Var220 and Var222

In [ ]:

```
data_new = data_new.drop(['Var220','Var222'], axis = 1)
```

In [ ]:

```
data_new.shape
```

Out[ ]:

```
(40000, 72)
```

In [ ]:

```
X_test_churn =  X_test_churn.drop(['Var220','Var222'], axis = 1)
```

In [ ]:

```
X_test_churn.shape
```

Out[ ]:

```
(10000, 72)
```

2 columns have been dropped from dataset. We're left with 72 features now instead of 74

Saving data in pickle file

In [ ]:

```
with open('X_train_churn.pickle', 'wb') as handle:
    pickle.dump(data_new, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [ ]:

```
with open('y_train_churn.pickle', 'wb') as handle:
    pickle.dump(y_train_churn, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [ ]:

```
with open('X_test_churn.pickle', 'wb') as handle:
    pickle.dump(X_test_churn, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [ ]:

```
with open('y_test_churn.pickle', 'wb') as handle:
    pickle.dump(y_test_churn, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [ ]:

```
with open('train_labels.pickle', 'wb') as handle:
    pickle.dump(train_labels, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [ ]:

```
with open('test_labels.pickle', 'wb') as handle:
    pickle.dump(test_labels, handle, protocol=pickle.HIGHEST_PROTOCOL)
```