Self Case Study - 1

Customer Relationship Prediction - Upselling

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 63.1 MB/s
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages (from scikit-lea
rn) (1.19.5)
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)
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: scikit-learn in /usr/local/lib/python3.7/dist-packages (from dython) (0.
24.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from dython) (1.19.5)
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)
Collecting scikit-plot>=0.3.7
  Downloading scikit_plot-0.3.7-py3-none-any.whl (33 kB)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (from dython) (3.2.
2)
Requirement already satisfied: pandas>=0.23.4 in /usr/local/lib/python3.7/dist-packages (from dython) (
1.1.5)
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: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplo
tlib->dython) (1.3.1)
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: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib-
>dython) (0.10.0)
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 484 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 [ ]:

```
upselling_labels = pd.read_csv('/content/drive/MyDrive/Case Study 1/Data/EDA/orange_small_train_upselli
ng.labels', header = None,names = ['Upselling'])
```

In [ ]:

```
upselling_labels.head()
```

Out[ ]:

	Upselling
0	-1
1	-1
2	-1
3	-1
4	-1

In [ ]:

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

In [ ]:

```
upselling_labels.shape
```

Out[ ]:

```
(50000, 1)
```

In [ ]:

```
upselling_labels.head()
```

Out[ ]:

	Upselling
0	0
1	0
2	0
3	0
4	0

Splitting data into train and test before data analysis

In [ ]:

```
X_train_upselling, X_test_upselling, y_train_upselling, y_test_upselling = train_test_split(data,upsell
ing_labels, test_size = 0.2, stratify = upselling_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 [ ]:

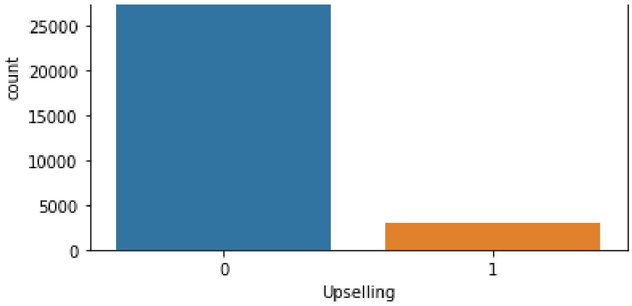
```
y_train_upselling.value_counts()
```

Out[ ]:

```
Upselling
0            37054
1             2946
dtype: int64
```

In [ ]:

```
plot_class_dist('Upselling', y_train_upselling)
```



Observation:

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

Counting total NaNs for each feature

In [ ]:

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

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

In [ ]:

```
all_nan_columns = np.array(X_train_upselling.columns[X_train_upselling.isna().sum() == X_train_upsellin
g.shape[0]])
```

In [ ]:

```
print('Number of features which do not countain NaNs:',(X_train_upselling.notna().sum() == X_train_upse
lling.shape[0]).sum())
```

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

In [ ]:

```
not_nan_columns = np.array(X_train_upselling.columns[X_train_upselling.notna().sum() == X_train_upselli
ng.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_upselling.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_upselling.columns:
    nan_count = X_train_upselling[i].isna().sum()
    nan_count_array.append(nan_count)
```

In [ ]:

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

In [ ]:

```python
print(x)
```

```
+----------+----------------+
| Features | Number of NaNs |
+----------+----------------+
|   Var1   |     39424      |
|   Var2   |     39018      |
|   Var3   |     39018      |
|   Var4   |     38743      |
|   Var5   |     38796      |
|   Var6   |      4450      |
|   Var7   |      4450      |
|   Var8   |     40000      |
|   Var9   |     39424      |
|  Var10   |     38796      |
|  Var11   |     39018      |
|  Var12   |     39544      |
|  Var13   |      4450      |
|  Var14   |     39018      |
|  Var15   |     40000      |
|  Var16   |     38796      |
|  Var17   |     38743      |
|  Var18   |     38743      |
|  Var19   |     38743      |
|  Var20   |     40000      |
|  Var21   |      4450      |
|  Var22   |      4019      |
|  Var23   |     38796      |
|  Var24   |      5807      |
|  Var25   |      4019      |
|  Var26   |     38796      |
|  Var27   |     38796      |
|  Var28   |      4021      |
```

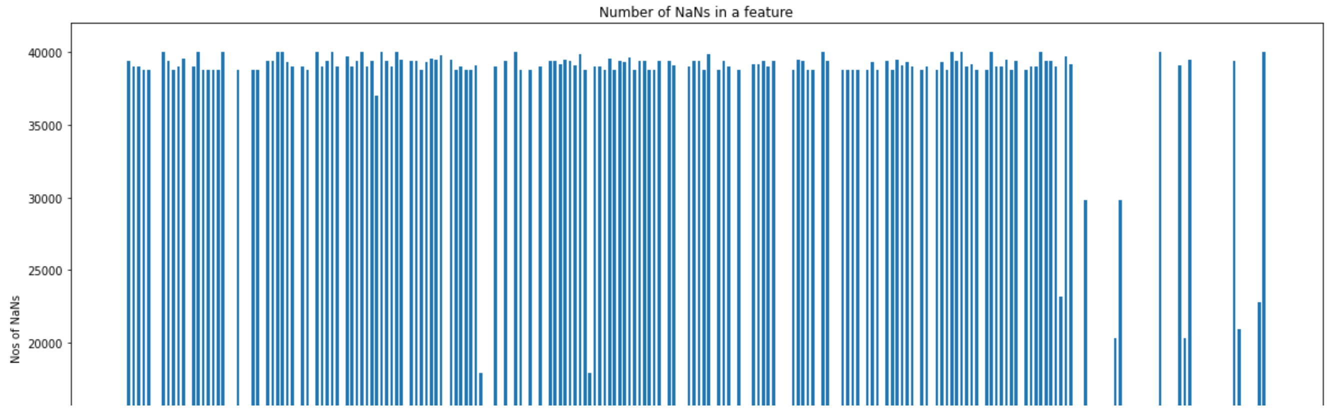
Var29	39424
Var30	39424
Var31	40000
Var32	40000
Var33	39330
Var34	39018
Var35	4019
Var36	39018
Var37	38743
Var38	4019
Var39	40000
Var40	39018
Var41	39424
Var42	40000
Var43	39018
Var44	4019
Var45	39731
Var46	39018
Var47	39424
Var48	40000
Var49	39018
Var50	39424
Var51	37005
Var52	40000
Var53	39424
Var54	39018
Var55	40000
Var56	39490
Var57	0
Var58	39424
Var59	39353
Var60	38796
Var61	39330
Var62	39544
Var63	39432
Var64	39804
Var65	4450
Var66	39432
Var67	38796
Var68	39018
Var69	38796
Var70	38796
Var71	39106
Var72	17912
Var73	0
Var74	4450
Var75	39018
Var76	4019
Var77	39424
Var78	4019
Var79	40000
Var80	38796
Var81	4450
Var82	38743
Var83	4019
Var84	39018
Var85	4019
Var86	39424
Var87	39424
Var88	39142
Var89	39490
Var90	39424
Var91	39106
Var92	39873
Var93	38796
Var94	17912
Var95	39018
Var96	39018
Var97	38796
Var98	39544
Var99	38743
Var100	39424
Var101	39315
Var102	39646
Var103	38796
Var104	39353
Var105	39353

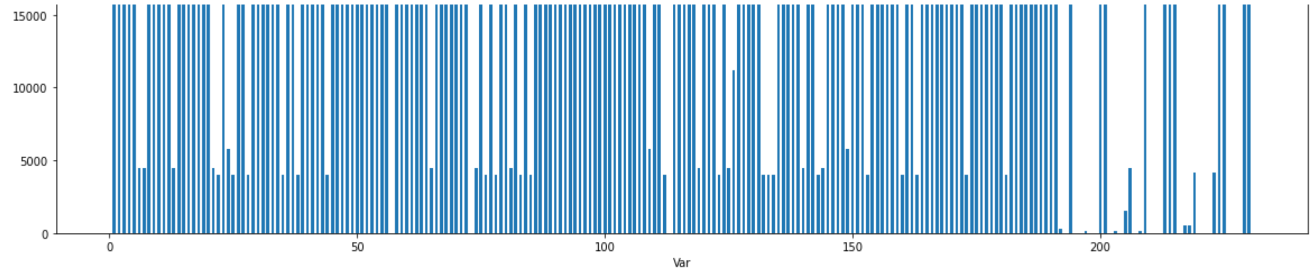
Var106	38743
Var107	38796
Var108	39424
Var109	5807
Var110	39424
Var111	39106
Var112	4019
Var113	0
Var114	39018
Var115	39353
Var116	39424
Var117	38743
Var118	39873
Var119	4450
Var120	38796
Var121	39424
Var122	39018
Var123	4019
Var124	38743
Var125	4450
Var126	11171
Var127	39142
Var128	39142
Var129	39424
Var130	39018
Var131	39424
Var132	4019
Var133	4019
Var134	4019
Var135	38743
Var136	39432
Var137	39424
Var138	38743
Var139	38796
Var140	4450
Var141	40000
Var142	39424
Var143	4019
Var144	4450
Var145	38743
Var146	38796
Var147	38796
Var148	38796
Var149	5807
Var150	38743
Var151	39330
Var152	38743
Var153	4019
Var154	39424
Var155	38743
Var156	39432
Var157	39106
Var158	39315
Var159	39018
Var160	4019
Var161	38743
Var162	39018
Var163	4019
Var164	38743
Var165	39315
Var166	38796
Var167	40000
Var168	39424
Var169	40000
Var170	39018
Var171	39142
Var172	38796
Var173	4019
Var174	38743
Var175	40000
Var176	39018
Var177	39018
Var178	39490
Var179	38743
Var180	39424
Var181	4019
Var182	38743

```
|  Var183  |     39018      |
|  Var184  |     39018      |
|  Var185  |     40000      |
|  Var186  |     39424      |
|  Var187  |     39424      |
|  Var188  |     39018      |
|  Var189  |     23200      |
|  Var190  |     39732      |
|  Var191  |     39142      |
|  Var192  |      295       |
|  Var193  |       0        |
|  Var194  |     29787      |
|  Var195  |       0        |
|  Var196  |       0        |
|  Var197  |      119       |
|  Var198  |       0        |
|  Var199  |       4        |
|  Var200  |     20319      |
|  Var201  |     29788      |
|  Var202  |       1        |
|  Var203  |      119       |
|  Var204  |       0        |
|  Var205  |      1543      |
|  Var206  |      4450      |
|  Var207  |       0        |
|  Var208  |      119       |
|  Var209  |     40000      |
|  Var210  |       0        |
|  Var211  |       0        |
|  Var212  |       0        |
|  Var213  |     39106      |
|  Var214  |     20319      |
|  Var215  |     39432      |
|  Var216  |       0        |
|  Var217  |      562       |
|  Var218  |      562       |
|  Var219  |      4167      |
|  Var220  |       0        |
|  Var221  |       0        |
|  Var222  |       0        |
|  Var223  |      4167      |
|  Var224  |     39353      |
|  Var225  |     20915      |
|  Var226  |       0        |
|  Var227  |       0        |
|  Var228  |       0        |
|  Var229  |     22808      |
|  Var230  |     40000      |
+----------+----------------+
```

In [ ]:

```
fig = plt.figure(figsize = (20, 10))
plt.bar(range(1,X_train_upselling.shape[1]+1), height = nan_count_array, width = 0.6,data = nan_count_a
rray)
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_upselling.iloc[:,190:].head()
```

Out[ ]:

	Var191	Var192	Var193	Var194	Var195	Var196	Var197	Var198	Var199	Var200	Var201	Var202
10317	NaN	LDPvyx7IEC	RO12	NaN	taul	1K8T	Bxva	60sg0bq	V_KvNzO	NaN	NaN	P3Gg
28605	NaN	1GdOj17ejg	RO12	NaN	taul	1K8T	TyGl	au1nqNs	7vJz3tk	NaN	NaN	2UUr
35917	NaN	crlgUHSK8h	RO12	SEuy	taul	1K8T	TyGl	WkHrLeh	NW71gM15FR	_YNrHue	smXZ	Mx5G
23069	NaN	FoxgUHSK8h	RO12	SEuy	taul	1K8T	AHgj	WI7JJYr	77i2xdu3Aa	F91wybA	smXZ	tF7g
36269	NaN	nTwTmBtueT	RO12	SEuy	taul	1K8T	0Xwj	47WdmQ4	PAsv0xf	6uM0zzl	smXZ	ZUB4

In [ ]:

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

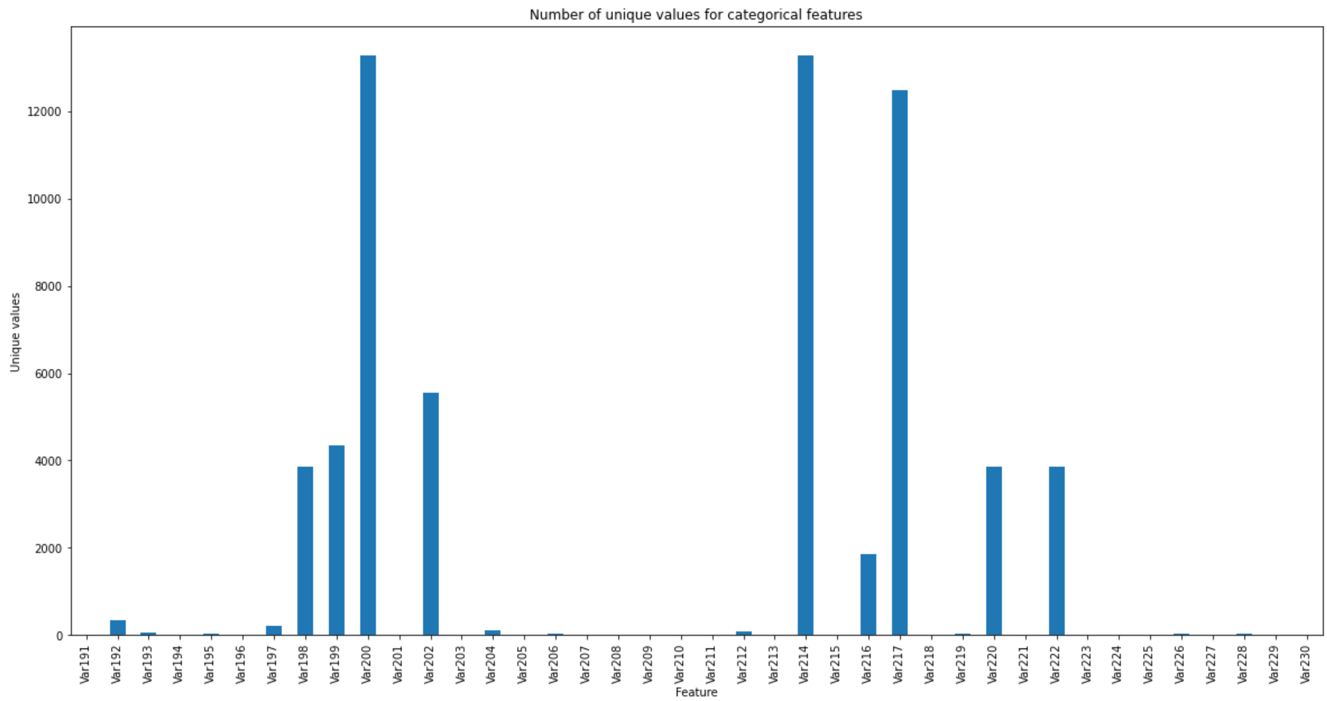
Out[ ]:

```
Var191        2
Var192      350
Var193       49
Var194        4
Var195       22
Var196        3
Var197      221
Var198     3863
Var199     4348
Var200    13291
Var201        3
Var202     5547
Var203        6
Var204      100
Var205        4
Var206       22
Var207       14
Var208        3
Var209        1
Var210        6
Var211        2
```

```
Var212        78
Var213         2
Var214     13291
Var215         2
Var216      1848
Var217     12476
Var218         3
Var219        23
Var220      3863
Var221         7
Var222      3863
Var223         5
Var224         2
Var225         4
Var226        23
Var227         7
Var228        30
Var229         5
Var230         1
dtype: int64
```

In [ ]:

```
#https://www.geeksforgeeks.org/how-to-count-distinct-values-of-a-pandas-dataframe-column/
fig = plt.figure(figsize = (20, 10))
X_train_upselling.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 [ ]:

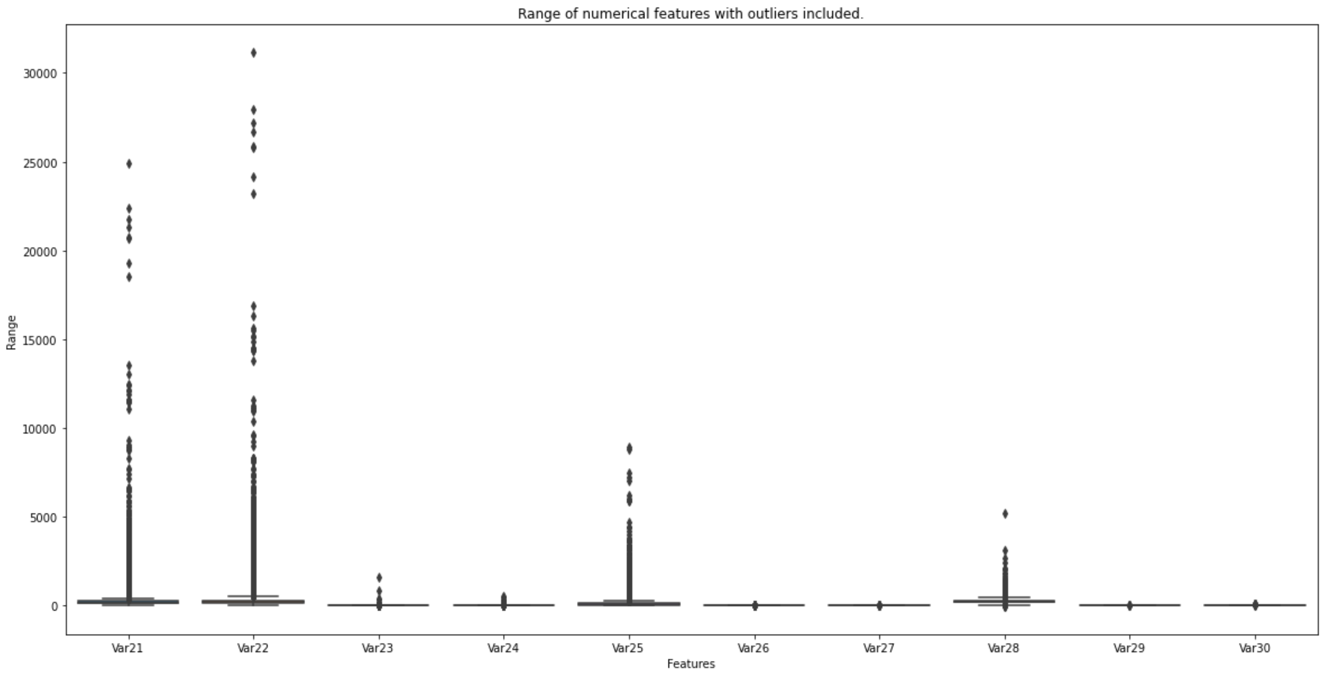
```
X_train_upselling.iloc[:,20:30].head()
```

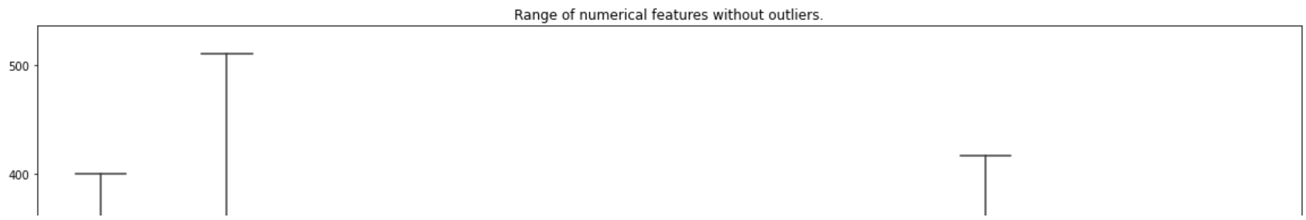
Out[ ]:

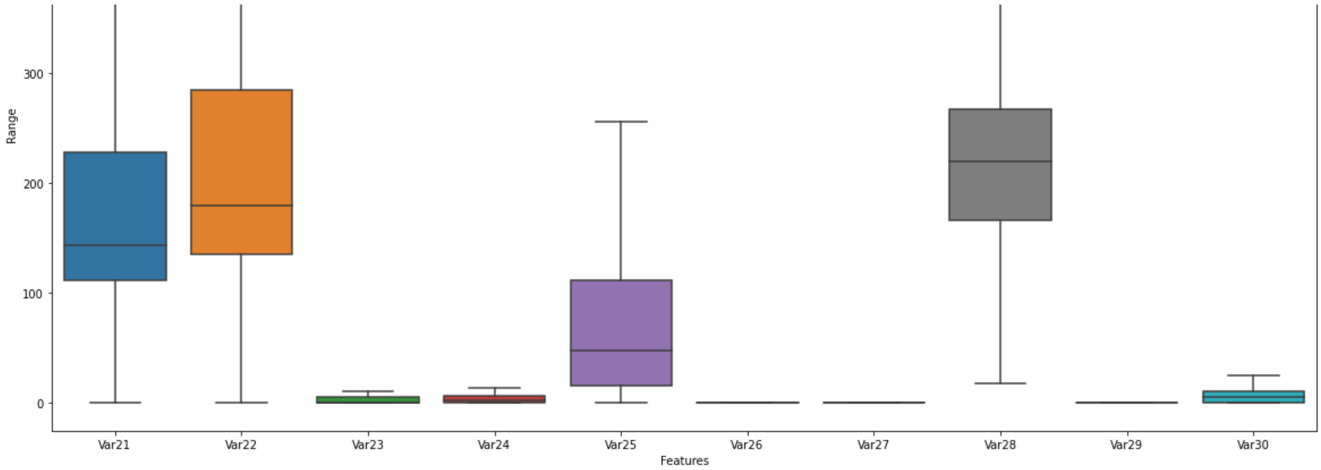
	Var21	Var22	Var23	Var24	Var25	Var26	Var27	Var28	Var29	Var30
10317	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
28605	96.0	120.0	NaN	0.0	0.0	NaN	NaN	186.64	NaN	NaN
35917	176.0	220.0	NaN	8.0	96.0	NaN	NaN	166.56	NaN	NaN
23069	136.0	170.0	NaN	0.0	32.0	NaN	NaN	253.52	NaN	NaN
36269	84.0	105.0	NaN	0.0	24.0	NaN	NaN	200.00	NaN	NaN

In [ ]:

```
# https://www.mikulskibartosz.name/how-to-remove-outliers-from-seaborn-boxplot-charts/
fig = plt.figure(figsize = (20, 10))
sns.boxplot(data = X_train_upselling.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_upselling.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_upselling,y_train_upselling],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.Upselling).sum().div(data_with_labels.Upselling.value_
counts(),axis = 0) * 100
```

Out[ ]:

	Var1	Var2	Var3	Var4	Var5	Var6	Var7	Var8	Var9	Var10	
Upselling											
0	98.485993	97.422680	97.422680	96.712905	96.901819	11.599288	11.545312	100.0	98.485993	96.901819	97
1	99.490835	99.083503	99.083503	98.676171	98.099117	5.159538	5.838425	100.0	99.490835	98.099117	99

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_upselling.drop(columns = all_nan_columns)
```

In [ ]:

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

In [ ]:

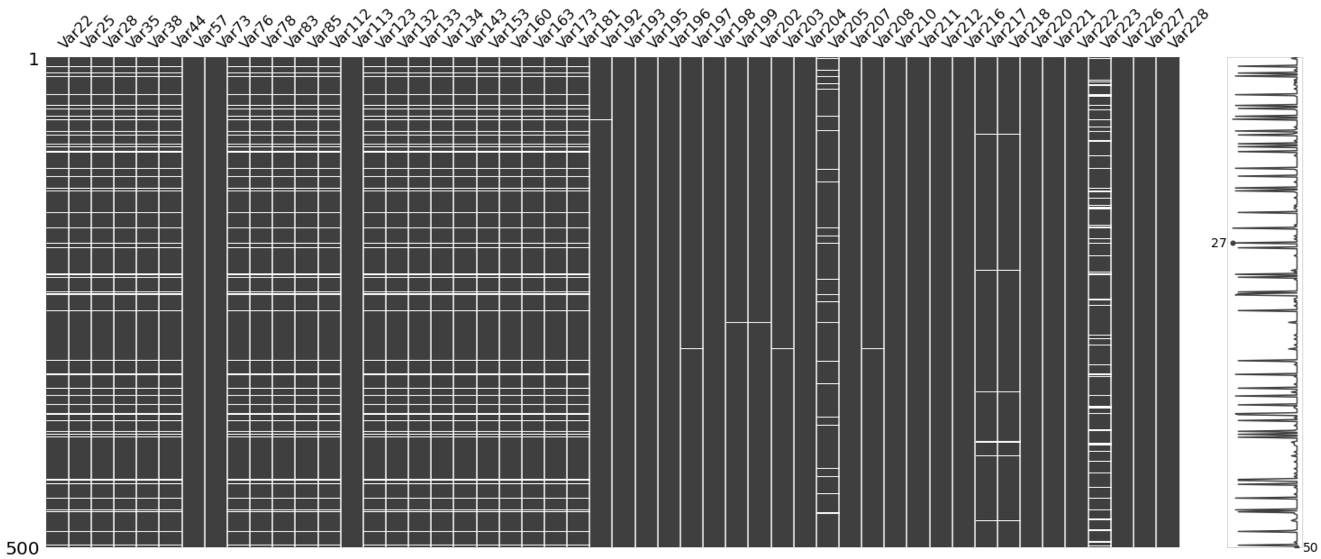
```
temp_data.shape
```

Out[ ]:

```
(40000, 212)
```

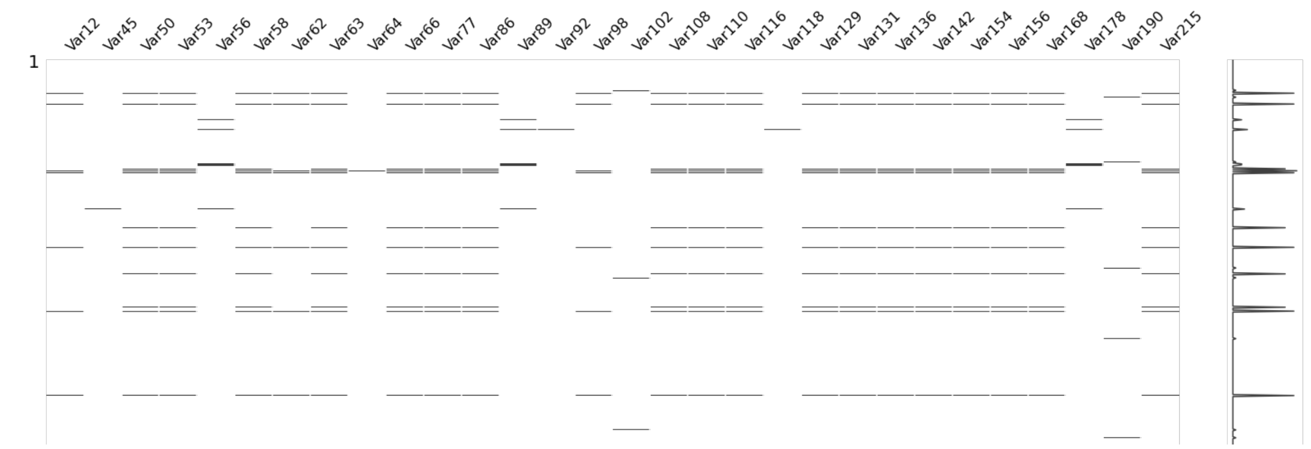
In [ ]:

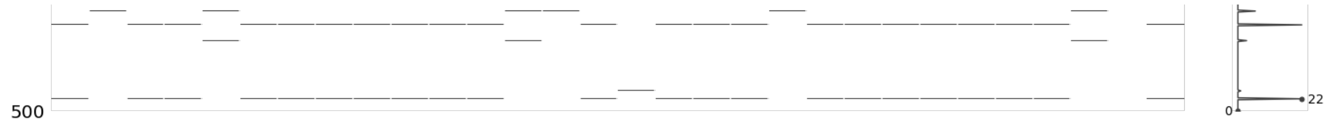
```
filter_data = msno.nullity_filter(temp_data, filter='top',n = 50)
msno.matrix(filter_data.sample(500))
plt.show()
```



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.

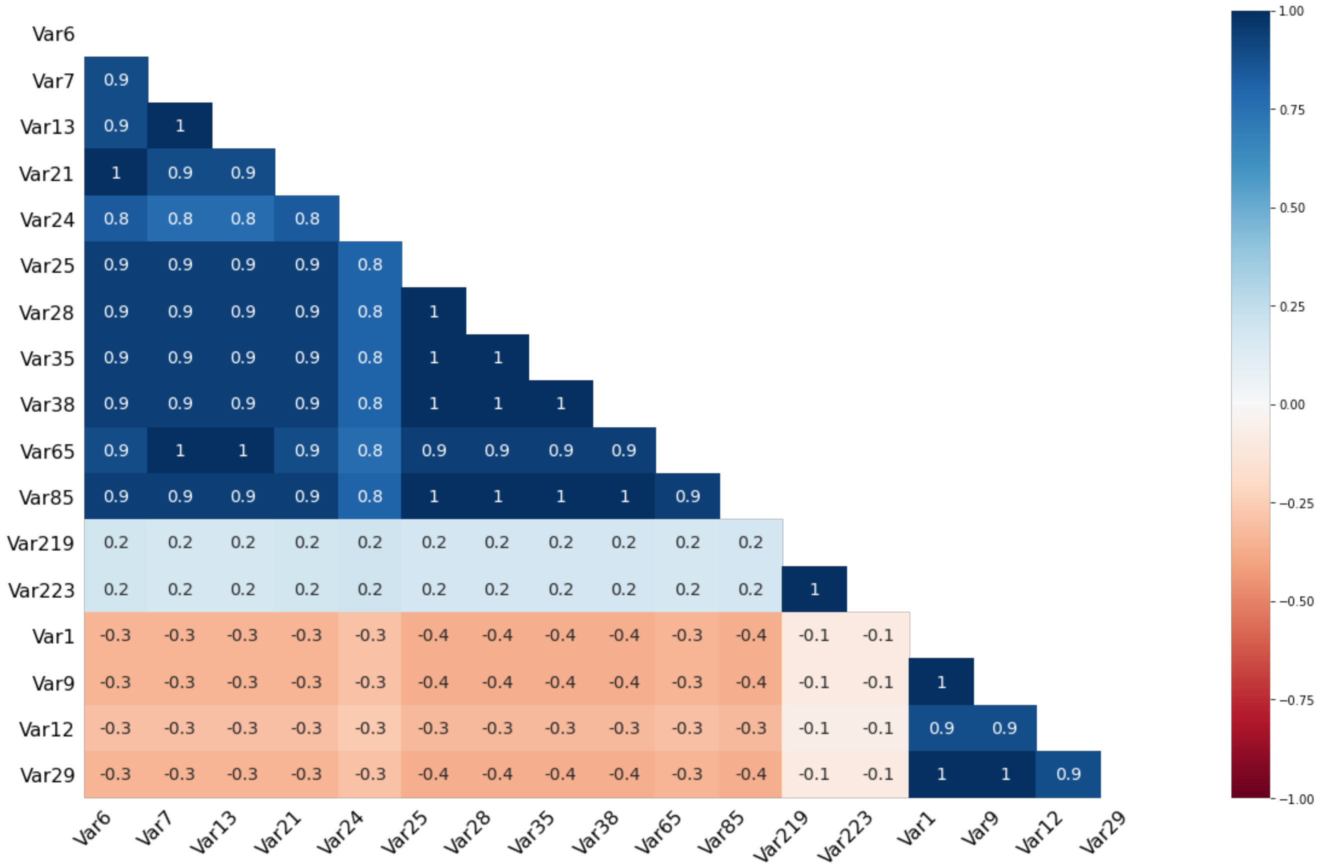
Checking for the correlation of missingness

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 0x7f5eba5d63d0>
```



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:

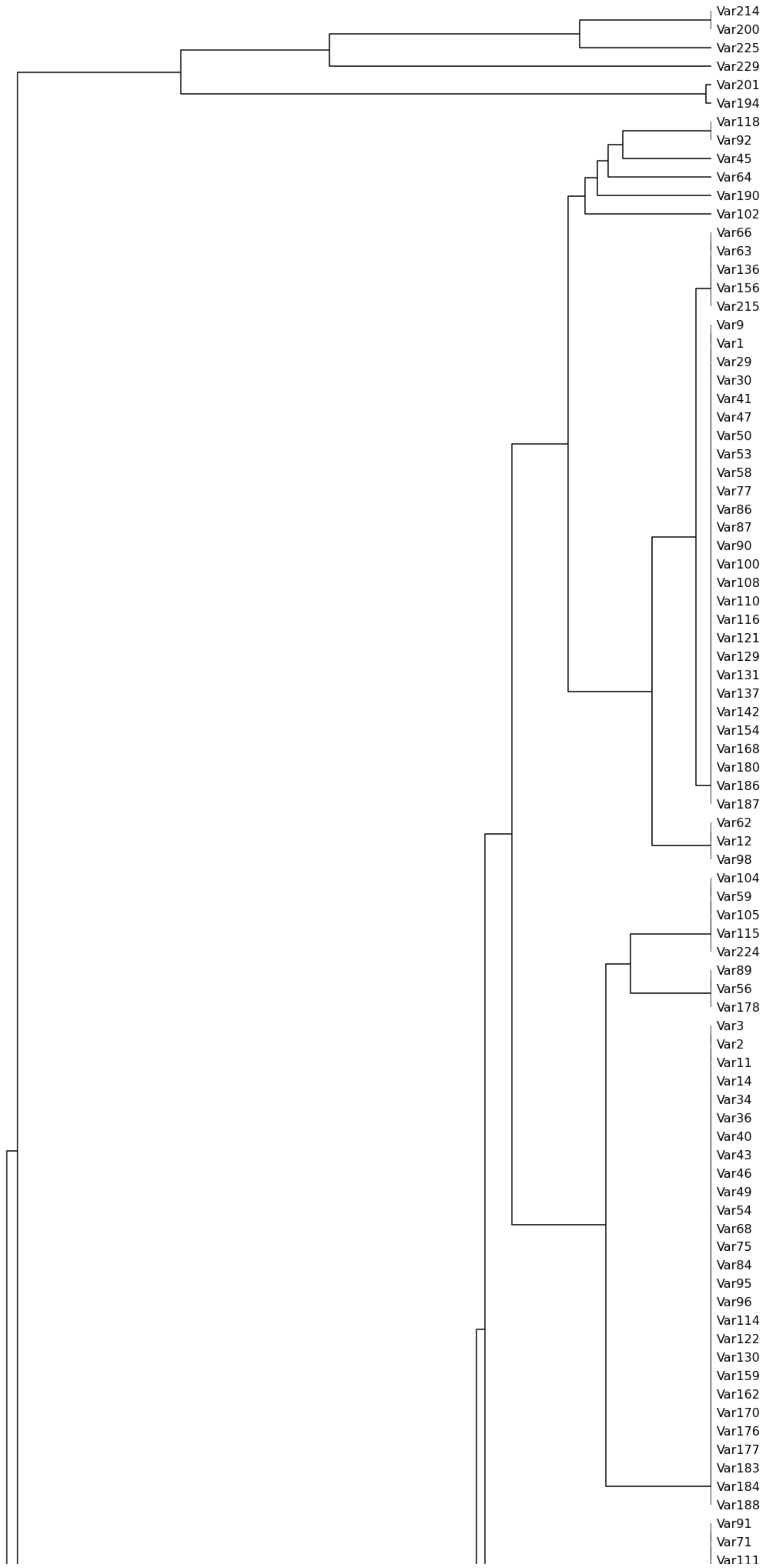
- As there is a correlation in missingness, we can rule out Missing Completely at random.

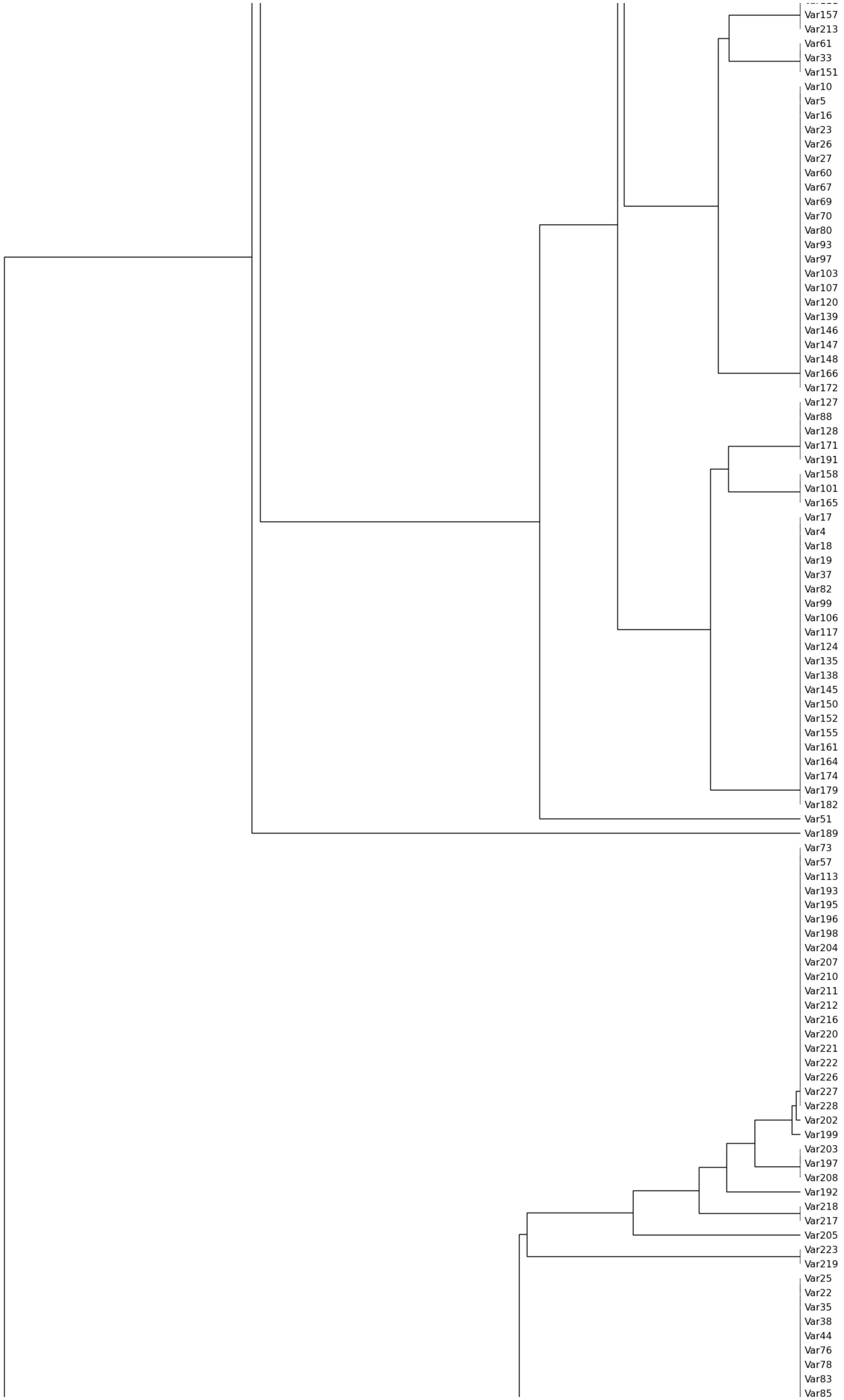
In [ ]:

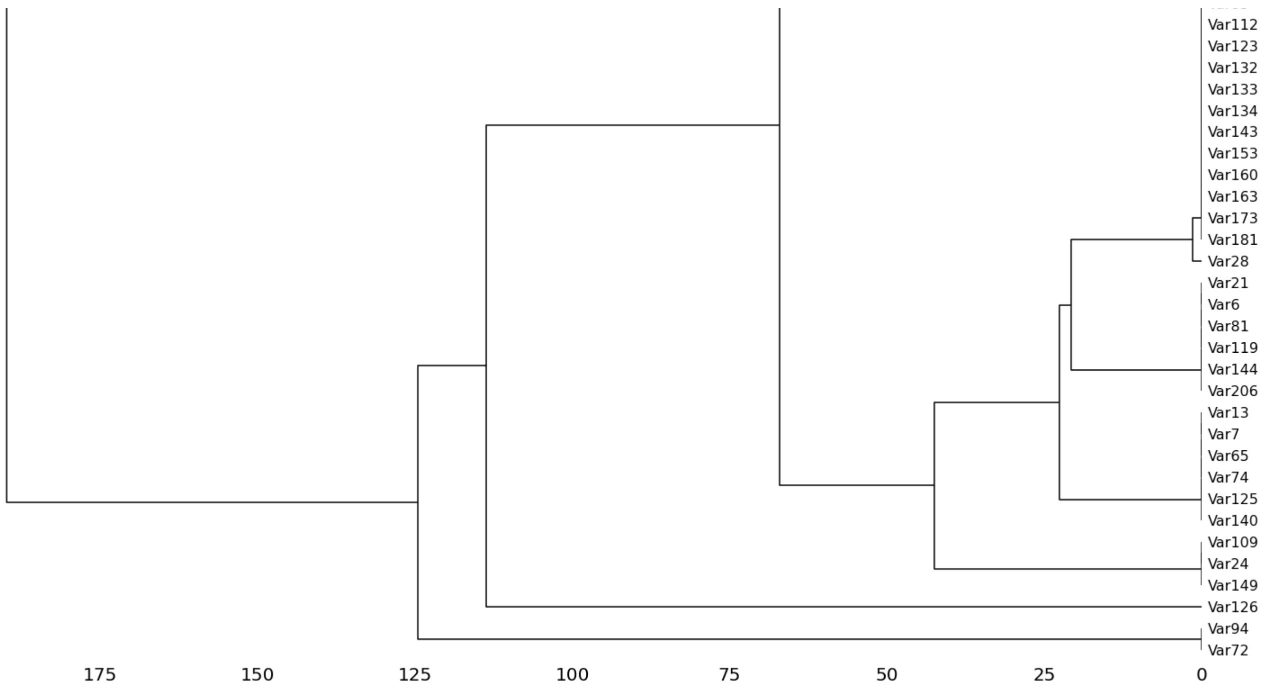
```
msno.dendrogram(temp_data)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5eb0fe9510>
```




Var157
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



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_upselling[all_cols]
```

In [ ]:

```
missing_data.head()
```

Out[ ]:

	Var123	Var132	Var133	Var143	Var153	Var160	Var163	Var173	Var181	Var193	Var195	Var196	Va
10317	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	RO12	taul	1K8T	60sg(
28605	72.0	0.0	2592000.0	0.0	7994680.0	12.0	0.0	0.0	0.0	RO12	taul	1K8T	au1n
35917	72.0	0.0	732350.0	0.0	5004120.0	8.0	1317846.0	0.0	0.0	RO12	taul	1K8T	WkHr
23069	114.0	0.0	161485.0	0.0	10573920.0	4.0	1036800.0	0.0	0.0	RO12	taul	1K8T	WI7J.
36269	6.0	0.0	76115.0	0.0	78508.0	30.0	0.0	0.0	0.0	RO12	taul	1K8T	47Wd

In [ ]:

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

Out[ ]:

```
10317     True
28605    False
35917    False
23069    False
36269    False
         ...
30492    False
33343    False
2267     False
18855    False
10263    False
Length: 40000, dtype: bool
```

In [ ]:

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

Out[ ]:

```
Var193       5
Var195       7
Var196       3
Var198    1135
Var204     100
Var207       6
Var210       5
Var211       2
Var212      15
Var216     151
Var220    1135
Var221       7
Var222    1135
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_upselling[all_cols]
```

In [ ]:

```
missing_data.head()
```

Out[ ]:

	Var123	Var132	Var133	Var143	Var153	Var160	Var163	Var173	Var181	Var57	Var73	Var113
10317	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.065798	8	-144605.2
28605	72.0	0.0	2592000.0	0.0	7994680.0	12.0	0.0	0.0	0.0	1.129246	32	169449.6
35917	72.0	0.0	732350.0	0.0	5004120.0	8.0	1317846.0	0.0	0.0	2.024567	52	-837464.0
23069	114.0	0.0	161485.0	0.0	10573920.0	4.0	1036800.0	0.0	0.0	0.555651	64	451612.0
36269	6.0	0.0	76115.0	0.0	78508.0	30.0	0.0	0.0	0.0	5.812006	96	-2035504.0

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       10704.0
Var132         160.0
Var133    15009900.0
```

```
Var143          18.0
Var153    13907800.0
Var160        4658.0
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     2.136296e-04
Var73     4.000000e+00
Var113   -9.684120e+06
dtype: float64
```

In [ ]:

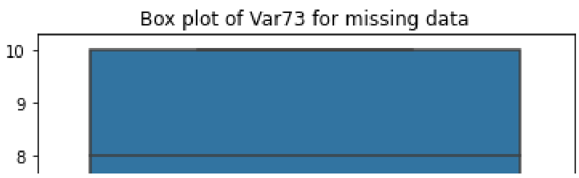
```
missing_data[missing_data.isna().any(axis =1)].max()
```

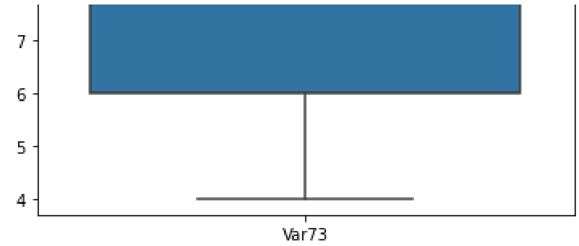
Out[ ]:

```
Var123             NaN
Var132             NaN
Var133             NaN
Var143             NaN
Var153             NaN
Var160             NaN
Var163             NaN
Var173             NaN
Var181             NaN
Var57     6.998932e+00
Var73     1.000000e+01
Var113    6.239680e+06
dtype: float64
```

In [ ]:

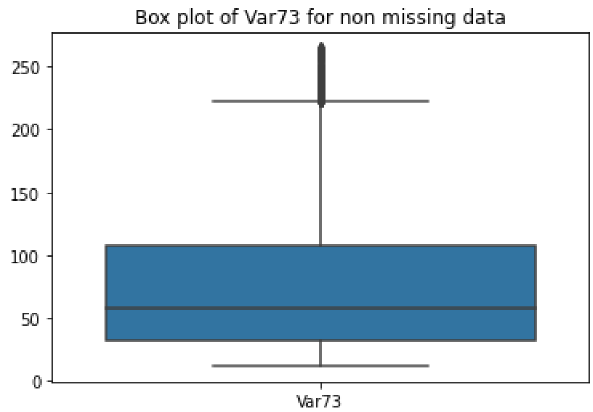
```
sns.boxplot(data = missing[['Var73']])
plt.title('Box plot of Var73 for missing data')
plt.show()
```





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_upselling.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_upselling.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_upselling.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_upselling.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_upselling.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_upselling.iloc[:,features]
data_new_test = X_test_upselling.iloc[:, features]
```

In [ ]:

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

In [ ]:

```
data_new.head()
```

Out[ ]:

	Var6	Var7	Var13	Var21	Var22	Var24	Var25	Var28	Var35	Var38	Var44	Var57	Var65	Var72	Var73
10317	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.065798	NaN	NaN	8
28605	602.0	7.0	0.0	96.0	120.0	0.0	0.0	186.64	0.0	5771202.0	0.0	1.129246	9.0	3.0	32
35917	2408.0	14.0	1004.0	176.0	220.0	8.0	96.0	166.56	0.0	1396482.0	0.0	2.024567	18.0	NaN	52
23069	924.0	7.0	64.0	136.0	170.0	0.0	32.0	253.52	0.0	9510660.0	0.0	0.555651	18.0	NaN	64
36269	483.0	7.0	544.0	84.0	105.0	0.0	24.0	200.00	0.0	1722.0	0.0	5.812006	9.0	3.0	96

In [ ]:

```
data_new_test.head()
```

Out[ ]:

	Var6	Var7	Var13	Var21	Var22	Var24	Var25	Var28	Var35	Var38	Var44	Var57	Var65	Var72	Var73
13186	1141.0	7.0	780.0	732.0	915.0	36.0	392.0	283.44	0.0	2759226.0	0.0	1.680624	9.0	3.0	150
49753	945.0	7.0	536.0	176.0	220.0	0.0	144.0	269.12	0.0	9150060.0	0.0	4.666951	18.0	3.0	40
34860	1456.0	7.0	1652.0	212.0	265.0	2.0	56.0	204.56	10.0	6866940.0	0.0	3.483230	9.0	3.0	84
4957	805.0	7.0	84.0	144.0	180.0	0.0	32.0	253.52	0.0	7906020.0	0.0	2.327708	9.0	3.0	32
43726	1106.0	7.0	4.0	164.0	205.0	4.0	88.0	308.08	0.0	2242956.0	0.0	1.941038	9.0	NaN	50

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=True);
```

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, 10317 to 10263
Data columns (total 74 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Var6    35550 non-null  float64
 1   Var7    35550 non-null  float64
 2   Var13   35550 non-null  float64
 3   Var21   35550 non-null  float64
 4   Var22   35981 non-null  float64
 5   Var24   34193 non-null  float64
 6   Var25   35981 non-null  float64
 7   Var28   35979 non-null  float64
 8   Var35   35981 non-null  float64
 9   Var38   35981 non-null  float64
 10  Var44   35981 non-null  float64
 11  Var57   40000 non-null  float64
 12  Var65   35550 non-null  float64
 13  Var72   22088 non-null  float64
 14  Var73   40000 non-null  int64  
 15  Var74   35550 non-null  float64
 16  Var76   35981 non-null  float64
 17  Var78   35981 non-null  float64
 18  Var81   35550 non-null  float64
 19  Var83   35981 non-null  float64
 20  Var85   35981 non-null  float64
 21  Var94   22088 non-null  float64
 22  Var109  34193 non-null  float64
 23  Var112  35981 non-null  float64
 24  Var113  40000 non-null  float64
 25  Var119  35550 non-null  float64
 26  Var123  35981 non-null  float64
 27  Var125  35550 non-null  float64
 28  Var126  28829 non-null  float64
 29  Var132  35981 non-null  float64
 30  Var133  35981 non-null  float64
 31  Var134  35981 non-null  float64
 32  Var140  35550 non-null  float64
 33  Var143  35981 non-null  float64
 34  Var144  35550 non-null  float64
 35  Var149  34193 non-null  float64
 36  Var153  35981 non-null  float64
 37  Var160  35981 non-null  float64
 38  Var163  35981 non-null  float64
 39  Var173  35981 non-null  float64
 40  Var181  35981 non-null  float64
 41  Var189  16800 non-null  float64
 42  Var192  39705 non-null  object 
 43  Var193  40000 non-null  object 
 44  Var195  40000 non-null  object 
 45  Var196  40000 non-null  object 
 46  Var197  39881 non-null  object 
 47  Var198  40000 non-null  object 
 48  Var199  39996 non-null  object 
 49  Var200  19681 non-null  object 
 50  Var202  39999 non-null  object 
 51  Var203  39881 non-null  object 
 52  Var204  40000 non-null  object 
 53  Var205  38457 non-null  object 
 54  Var206  35550 non-null  object 
 55  Var207  40000 non-null  object 
```

```
 56  Var208  39881 non-null  object
 57  Var210  40000 non-null  object
 58  Var211  40000 non-null  object
 59  Var212  40000 non-null  object
 60  Var214  19681 non-null  object
 61  Var216  40000 non-null  object
 62  Var217  39438 non-null  object
 63  Var218  39438 non-null  object
 64  Var219  35833 non-null  object
 65  Var220  40000 non-null  object
 66  Var221  40000 non-null  object
 67  Var222  40000 non-null  object
 68  Var223  35833 non-null  object
 69  Var225  19085 non-null  object
 70  Var226  40000 non-null  object
 71  Var227  40000 non-null  object
 72  Var228  40000 non-null  object
 73  Var229  17192 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
10317	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.065798	NaN	NaN	8
28605	602.0	7.0	0.0	96.0	120.0	0.0	0.0	186.64	0.0	5771202.0	0.0	1.129246	9.0	3.0	32
35917	2408.0	14.0	1004.0	176.0	220.0	8.0	96.0	166.56	0.0	1396482.0	0.0	2.024567	18.0	NaN	52
23069	924.0	7.0	64.0	136.0	170.0	0.0	32.0	253.52	0.0	9510660.0	0.0	0.555651	18.0	NaN	64
36269	483.0	7.0	544.0	84.0	105.0	0.0	24.0	200.00	0.0	1722.0	0.0	5.812006	9.0	3.0	96

In [ ]:

```
numerical_data_test.head()
```

Out[ ]:

	Var6	Var7	Var13	Var21	Var22	Var24	Var25	Var28	Var35	Var38	Var44	Var57	Var65	Var72	Var73
13186	1141.0	7.0	780.0	732.0	915.0	36.0	392.0	283.44	0.0	2759226.0	0.0	1.680624	9.0	3.0	150
49753	945.0	7.0	536.0	176.0	220.0	0.0	144.0	269.12	0.0	9150060.0	0.0	4.666951	18.0	3.0	40
34860	1456.0	7.0	1652.0	212.0	265.0	2.0	56.0	204.56	10.0	6866940.0	0.0	3.483230	9.0	3.0	84
4957	805.0	7.0	84.0	144.0	180.0	0.0	32.0	253.52	0.0	7906020.0	0.0	2.327708	9.0	3.0	32
43726	1106.0	7.0	4.0	164.0	205.0	4.0	88.0	308.08	0.0	2242956.0	0.0	1.941038	9.0	NaN	50

In [ ]:

```
numerical_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 40000 entries, 10317 to 10263
Data columns (total 42 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Var6     35550 non-null  float64
 1   Var7     35550 non-null  float64
 2   Var13    35550 non-null  float64
 3   Var21    35550 non-null  float64
 4   Var22    35981 non-null  float64
 5   Var24    34193 non-null  float64
 6   Var25    35981 non-null  float64
 7   Var28    35979 non-null  float64
 8   Var35    35981 non-null  float64
 9   Var38    35981 non-null  float64
 10  Var44    35981 non-null  float64
 11  Var57    40000 non-null  float64
 12  Var65    35550 non-null  float64
 13  Var72    22088 non-null  float64
 14  Var73    40000 non-null  int64  
 15  Var74    35550 non-null  float64
 16  Var76    35981 non-null  float64
 17  Var78    35981 non-null  float64
 18  Var81    35550 non-null  float64
 19  Var83    35981 non-null  float64
 20  Var85    35981 non-null  float64
 21  Var94    22088 non-null  float64
 22  Var109   34193 non-null  float64
 23  Var112   35981 non-null  float64
 24  Var113   40000 non-null  float64
 25  Var119   35550 non-null  float64
 26  Var123   35981 non-null  float64
 27  Var125   35550 non-null  float64
 28  Var126   28829 non-null  float64
 29  Var132   35981 non-null  float64
 30  Var133   35981 non-null  float64
 31  Var134   35981 non-null  float64
 32  Var140   35550 non-null  float64
 33  Var143   35981 non-null  float64
 34  Var144   35550 non-null  float64
 35  Var149   34193 non-null  float64
 36  Var153   35981 non-null  float64
 37  Var160   35981 non-null  float64
 38  Var163   35981 non-null  float64
 39  Var173   35981 non-null  float64
 40  Var181   35981 non-null  float64
 41  Var189   16800 non-null  float64
dtypes: float64(41), int64(1)
memory usage: 13.1 MB
```

In [ ]:

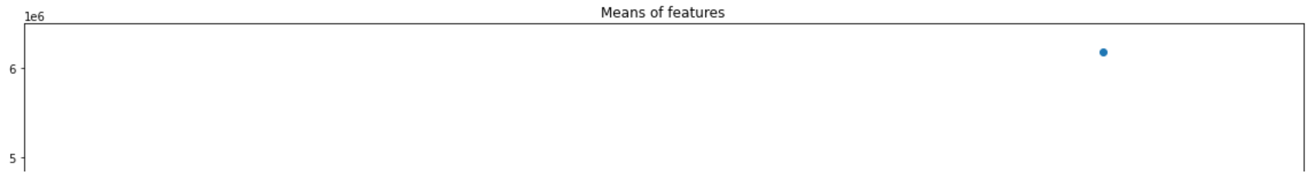
```
means = numerical_data.mean()
```

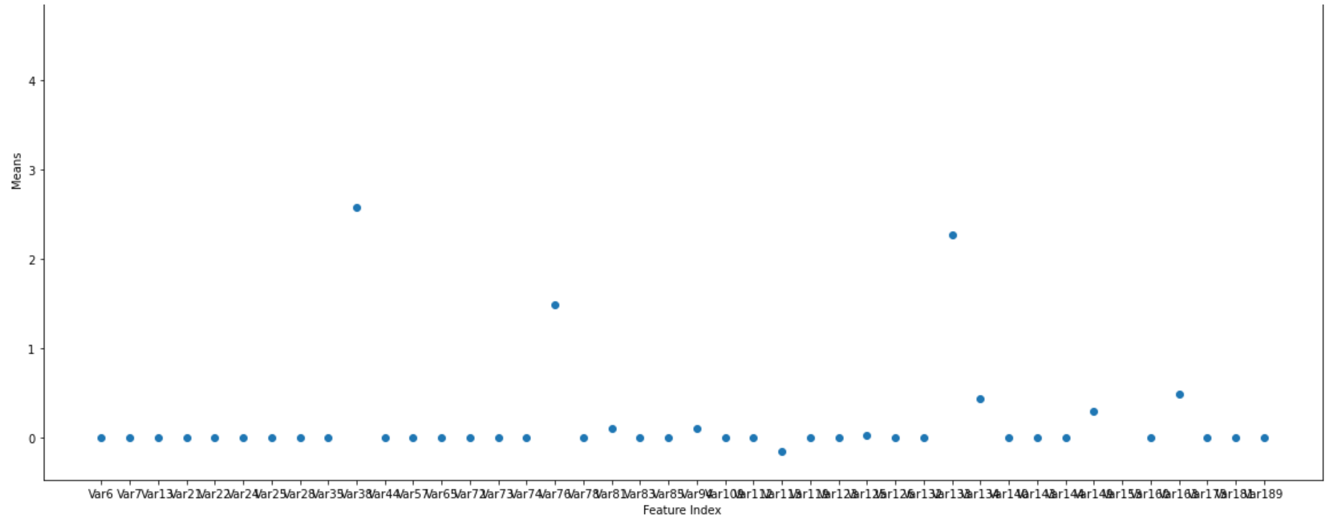
In [ ]:

```
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 [ ]:

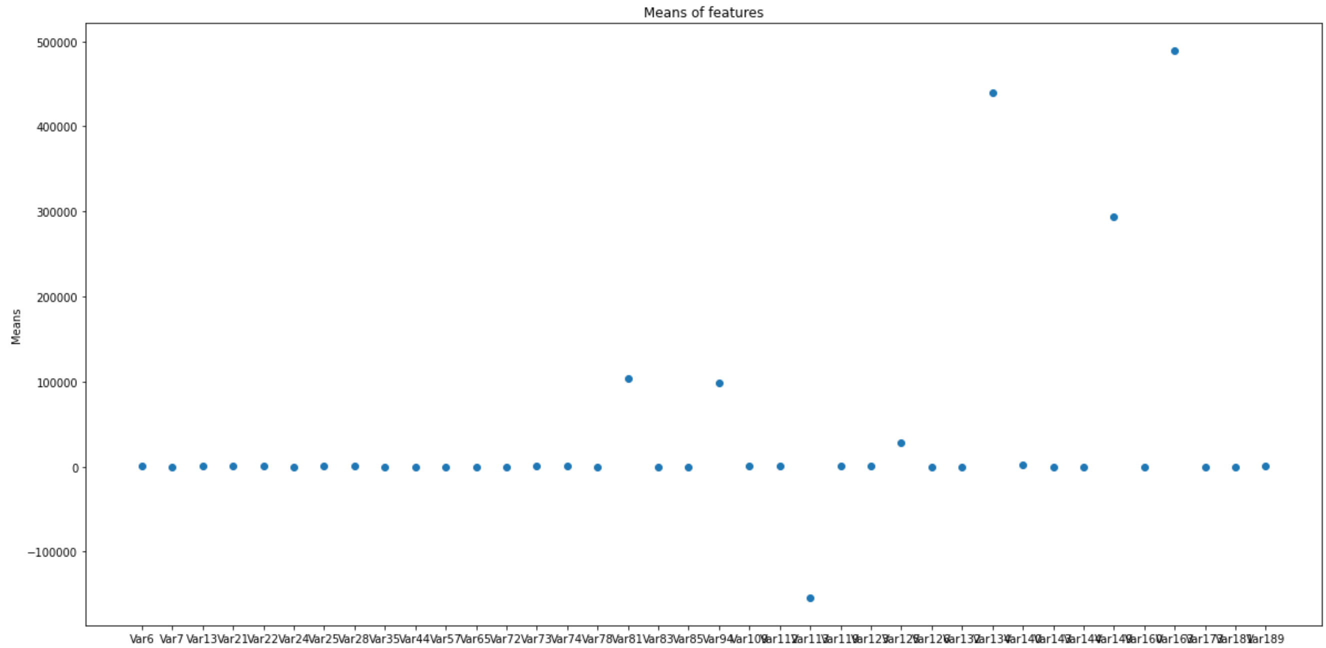
```
filter_means.shape
```

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 [ ]:

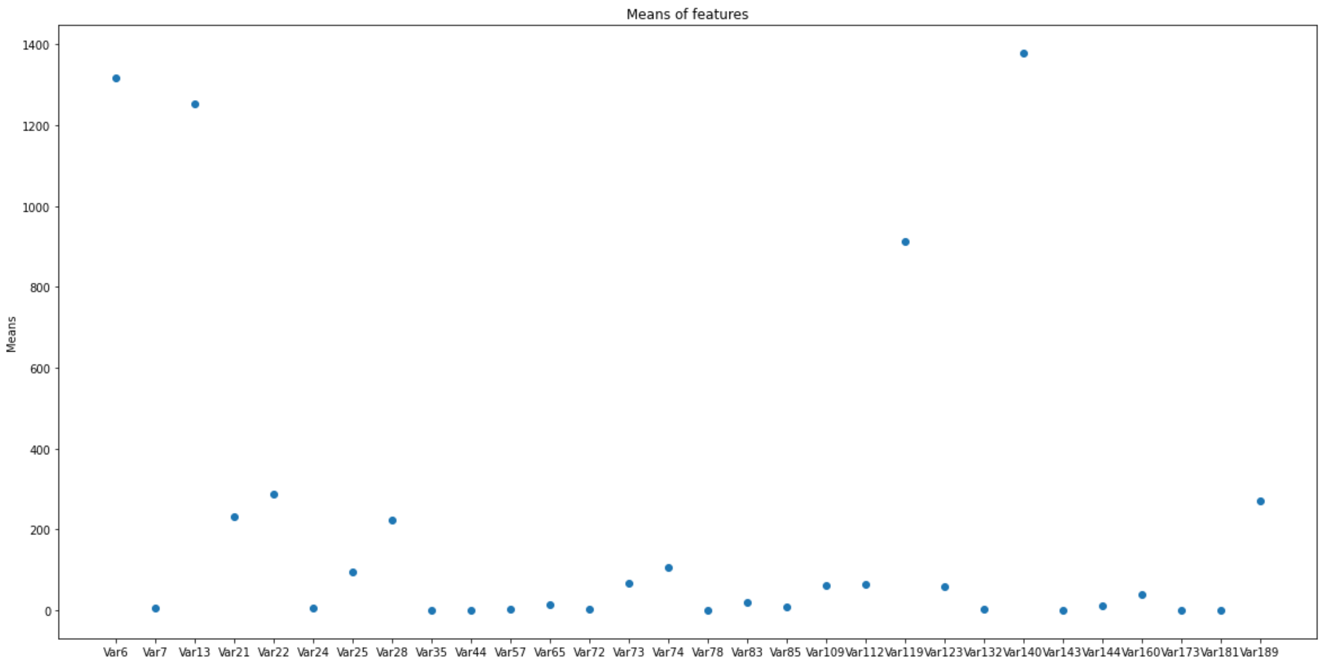
```
filter_means.shape
```

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 [ ]:

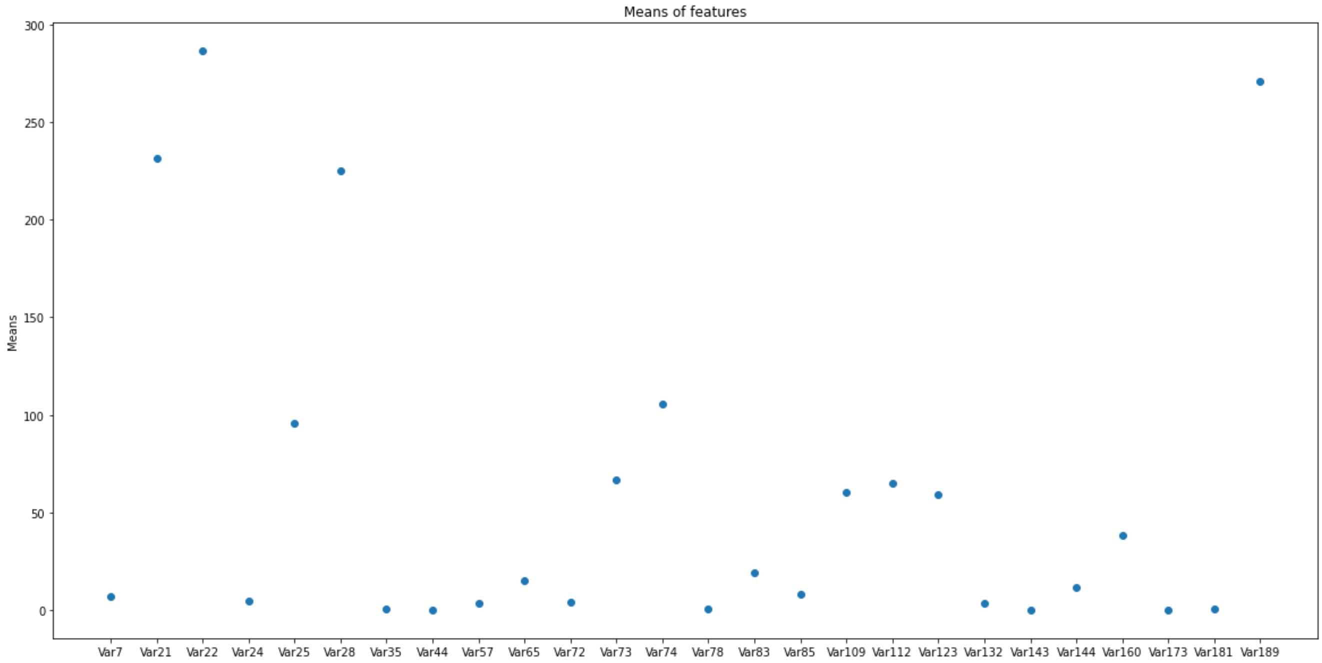
```
filter_means.shape
```

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 [ ]:

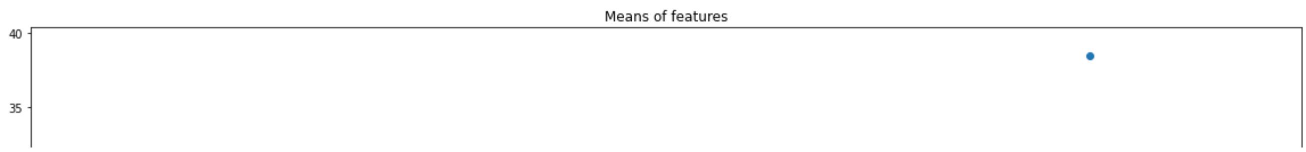
```
filter_means.shape
```

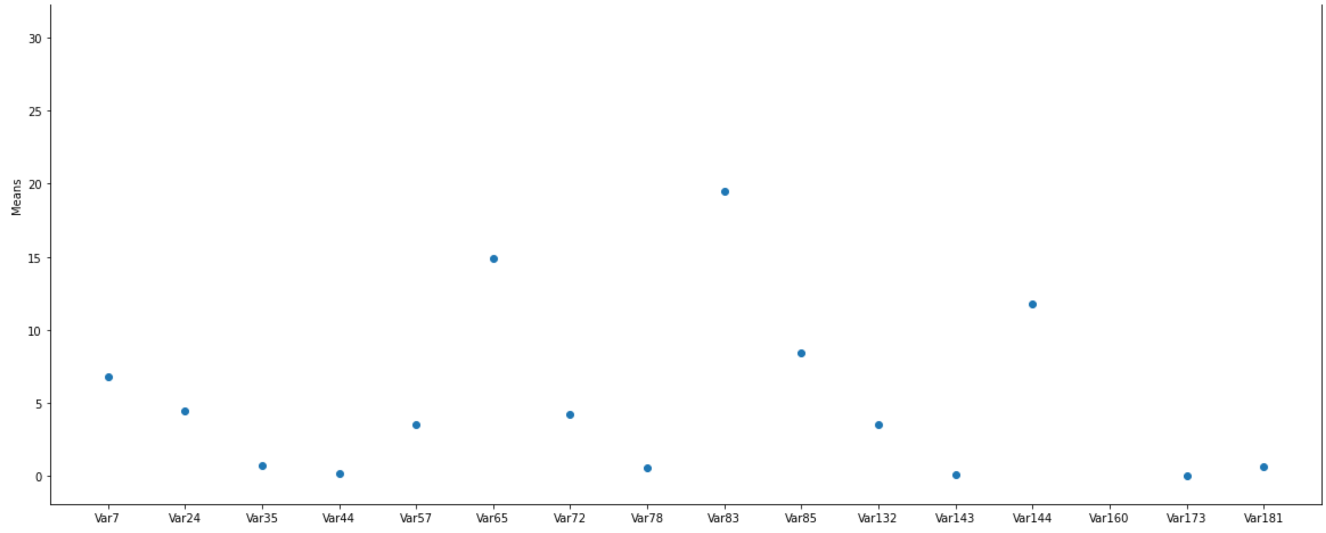
Out[ ]:

```
(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      1.367001e+03
Var7      6.773763e+00
Var13     1.234362e+03
Var21     2.465096e+02
Var22     3.054279e+02
Var24     4.640550e+00
Var25     1.005416e+02
Var28     2.237658e+02
Var35     7.513873e-01
Var38     2.612301e+06
Var44     1.698113e-01
Var57     3.503533e+00
Var65     1.475491e+01
Var72     4.143709e+00
Var73     6.688360e+01
Var74     9.580990e+01
Var76     1.507221e+06
Var78     5.587125e-01
Var81     1.035305e+05
Var83     2.224750e+01
Var85     8.668368e+00
Var94     9.837957e+04
```

```
Var94     9.837957e+04
Var109    6.306447e+01
Var112    7.110322e+01
Var113   -1.485999e+05
Var119    9.289054e+02
Var123    6.488990e+01
Var125    2.782753e+04
Var126   -6.925941e-01
Var132    3.651942e+00
Var133    2.284673e+06
Var134    4.274692e+05
Var140    1.390349e+03
Var143    5.527192e-02
Var144    1.163614e+01
Var149    3.009567e+05
Var153    6.203112e+06
Var160    4.006260e+01
Var163    4.731397e+05
Var173    7.769145e-03
Var181    6.160932e-01
Var189    2.686329e+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 [ ]:

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

In [ ]:

```
data_new.head()
```

Out[ ]:

	Var6	Var7	Var13	Var21	Var22	Var24	Var25	Var28	Var35	Var38	Var44	Var57	Var65	Var72	Var73
10317	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.065798	NaN	NaN	8
28605	602.0	7.0	0.0	96.0	120.0	0.0	0.0	186.64	0.0	5771202.0	0.0	1.129246	9.0	3.0	32
35917	2408.0	14.0	1004.0	176.0	220.0	8.0	96.0	166.56	0.0	1396482.0	0.0	2.024567	18.0	NaN	52
23069	924.0	7.0	64.0	136.0	170.0	0.0	32.0	253.52	0.0	9510660.0	0.0	0.555651	18.0	NaN	64
36269	483.0	7.0	544.0	84.0	105.0	0.0	24.0	200.00	0.0	1722.0	0.0	5.812006	9.0	3.0	96

In [ ]:

```
data_new_test.head()
```

Out[ ]:

	Var6	Var7	Var13	Var21	Var22	Var24	Var25	Var28	Var35	Var38	Var44	Var57	Var65	Var72	Var73
13186	1141.0	7.0	780.0	732.0	915.0	36.0	392.0	283.44	0.0	2759226.0	0.0	1.680624	9.0	3.0	150
49753	945.0	7.0	536.0	176.0	220.0	0.0	144.0	269.12	0.0	9150060.0	0.0	4.666951	18.0	3.0	40
34860	1456.0	7.0	1652.0	212.0	265.0	2.0	56.0	204.56	10.0	6866940.0	0.0	3.483230	9.0	3.0	84
4957	805.0	7.0	84.0	144.0	180.0	0.0	32.0	253.52	0.0	7906020.0	0.0	2.327708	9.0	3.0	32
43726	1106.0	7.0	4.0	164.0	205.0	4.0	88.0	308.08	0.0	2242956.0	0.0	1.941038	9.0	NaN	50

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 [ ]:

```
data_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 40000 entries, 10317 to 10263
Data columns (total 74 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Var6    35550 non-null  float64
 1   Var7    35550 non-null  float64
 2   Var13   35550 non-null  float64
```

```
 3   Var21   35550 non-null  float64
 4   Var22   35981 non-null  float64
 5   Var24   34193 non-null  float64
 6   Var25   35981 non-null  float64
 7   Var28   35979 non-null  float64
 8   Var35   35981 non-null  float64
 9   Var38   35981 non-null  float64
 10  Var44   35981 non-null  float64
 11  Var57   40000 non-null  float64
 12  Var65   35550 non-null  float64
 13  Var72   22088 non-null  float64
 14  Var73   40000 non-null  int64
 15  Var74   35550 non-null  float64
 16  Var76   35981 non-null  float64
 17  Var78   35981 non-null  float64
 18  Var81   35550 non-null  float64
 19  Var83   35981 non-null  float64
 20  Var85   35981 non-null  float64
 21  Var94   22088 non-null  float64
 22  Var109  34193 non-null  float64
 23  Var112  35981 non-null  float64
 24  Var113  40000 non-null  float64
 25  Var119  35550 non-null  float64
 26  Var123  35981 non-null  float64
 27  Var125  35550 non-null  float64
 28  Var126  28829 non-null  float64
 29  Var132  35981 non-null  float64
 30  Var133  35981 non-null  float64
 31  Var134  35981 non-null  float64
 32  Var140  35550 non-null  float64
 33  Var143  35981 non-null  float64
 34  Var144  35550 non-null  float64
 35  Var149  34193 non-null  float64
 36  Var153  35981 non-null  float64
 37  Var160  35981 non-null  float64
 38  Var163  35981 non-null  float64
 39  Var173  35981 non-null  float64
 40  Var181  35981 non-null  float64
 41  Var189  16800 non-null  float64
 42  Var192  39705 non-null  object
 43  Var193  40000 non-null  object
 44  Var195  40000 non-null  object
 45  Var196  40000 non-null  object
 46  Var197  39881 non-null  object
 47  Var198  40000 non-null  object
 48  Var199  39996 non-null  object
 49  Var200  19681 non-null  object
 50  Var202  39999 non-null  object
 51  Var203  39881 non-null  object
 52  Var204  40000 non-null  object
 53  Var205  38457 non-null  object
 54  Var206  35550 non-null  object
 55  Var207  40000 non-null  object
 56  Var208  39881 non-null  object
 57  Var210  40000 non-null  object
 58  Var211  40000 non-null  object
 59  Var212  40000 non-null  object
 60  Var214  19681 non-null  object
 61  Var216  40000 non-null  object
 62  Var217  39438 non-null  object
 63  Var218  39438 non-null  object
 64  Var219  35833 non-null  object
 65  Var220  40000 non-null  object
 66  Var221  40000 non-null  object
 67  Var222  40000 non-null  object
 68  Var223  35833 non-null  object
 69  Var225  19085 non-null  object
 70  Var226  40000 non-null  object
 71  Var227  40000 non-null  object
 72  Var228  40000 non-null  object
 73  Var229  17192 non-null  object
dtypes: float64(41), int64(1), object(32)
memory usage: 22.9+ MB
```

In [ ]:

```
data_new.mean()
```

Out[ ]:

```
Var6      1.316258e+03
Var7      6.818453e+00
Var13     1.253530e+03
Var21     2.315091e+02
Var22     2.864435e+02
Var24     4.474659e+00
Var25     9.589683e+01
Var28     2.246934e+02
Var35     7.081515e-01
Var38     2.570795e+06
Var44     1.660877e-01
Var57     3.514506e+00
Var65     1.489747e+01
Var72     4.202418e+00
Var73     6.658045e+01
Var74     1.056254e+02
Var76     1.485880e+06
Var78     5.286957e-01
Var81     1.029720e+05
Var83     1.946666e+01
Var85     8.409105e+00
Var94     9.874407e+04
Var109    6.034288e+01
Var112    6.499853e+01
Var113   -1.544483e+05
Var119    9.129018e+02
Var123    5.901064e+01
Var125    2.790269e+04
Var126   -5.189913e-01
Var132    3.492732e+00
Var133    2.270792e+06
Var134    4.398122e+05
Var140    1.378981e+03
Var143    5.869765e-02
Var144    1.175063e+01
Var149    2.934068e+05
Var153    6.176672e+06
Var160    3.848759e+01
Var163    4.893178e+05
Var173    6.614602e-03
Var181    6.102943e-01
Var189    2.705214e+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	
10317	1316.257947	6.818453	1253.530127	231.509086	286.44354	4.474659	95.896834	224.693449	0.708152	2.570795
28605	602.000000	7.000000	0.000000	96.000000	120.00000	0.000000	0.000000	186.640000	0.000000	5.771202
35917	2408.000000	14.000000	1004.000000	176.000000	220.00000	8.000000	96.000000	166.560000	0.000000	1.396482
23069	924.000000	7.000000	64.000000	136.000000	170.00000	0.000000	32.000000	253.520000	0.000000	9.51066
36269	483.000000	7.000000	544.000000	84.000000	105.00000	0.000000	24.000000	200.000000	0.000000	1.72200

In [ ]:

```
data_new_imputed_test.head()
```

Out[ ]:

	Var6	Var7	Var13	Var21	Var22	Var24	Var25	Var28	Var35	Var38	Var44	Var57	Var65	Var72	Var7
13186	1141.0	7.0	780.0	732.0	915.0	36.0	392.0	283.44	0.0	2759226.0	0.0	1.680624	9.0	3.000000	150
49753	945.0	7.0	536.0	176.0	220.0	0.0	144.0	269.12	0.0	9150060.0	0.0	4.666951	18.0	3.000000	40
34860	1456.0	7.0	1652.0	212.0	265.0	2.0	56.0	204.56	10.0	6866940.0	0.0	3.483230	9.0	3.000000	84
4957	805.0	7.0	84.0	144.0	180.0	0.0	32.0	253.52	0.0	7906020.0	0.0	2.327708	9.0	3.000000	32
43726	1106.0	7.0	4.0	164.0	205.0	4.0	88.0	308.08	0.0	2242956.0	0.0	1.941038	9.0	4.143709	50

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	Var202	Var203	Var204	Var2
10317	LDPvyx7IEC	RO12	taul	1K8T	Bxva	60sg0bq	V_KvNzO	Others	P3Gg	9_Y1	YGOC	VpdC
28605	1GdOj17ejg	RO12	taul	1K8T	TyGl	au1nqNs	7vJz3tk	Others	2UUr	9_Y1	zfpA	VpdC
35917	crlgUHSK8h	RO12	taul	1K8T	TyGl	WkHrLeh	NW71gM15FR	_YNrHue	Mx5G	9_Y1	RVjC	VpdC
23069	FoxgUHSK8h	RO12	taul	1K8T	AHgj	Wl7JJYr	77i2xdu3Aa	F91wybA	tF7g	9_Y1	z5Ry	09_Q
36269	nTwTmBtueT	RO12	taul	1K8T	0Xwj	47WdmQ4	PAsv0xf	6uM0zzl	ZUB4	HLqf	47ra	sJzTl

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 Desc	Var198 feature1	Var222 feature2
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
21499	NldASpP	JFM1BiF	NKv4yOc
4878	NldASpP	JFM1BiF	NKv4yOc
33091	NldASpP	JFM1BiF	NKv4yOc
33827	NldASpP	JFM1BiF	NKv4yOc
16700	NldASpP	JFM1BiF	NKv4yOc
41830	NldASpP	JFM1BiF	NKv4yOc
37386	NldASpP	JFM1BiF	NKv4yOc
29914	NldASpP	JFM1BiF	NKv4yOc
8274	NldASpP	JFM1BiF	NKv4yOc
36636	NldASpP	JFM1BiF	NKv4yOc
12170	NldASpP	JFM1BiF	NKv4yOc
40704	NldASpP	JFM1BiF	NKv4yOc
38508	NldASpP	JFM1BiF	NKv4yOc
21672	NldASpP	JFM1BiF	NKv4yOc
36377	NldASpP	JFM1BiF	NKv4yOc
39116	NldASpP	JFM1BiF	NKv4yOc
46360	NldASpP	JFM1BiF	NKv4yOc
16980	NldASpP	JFM1BiF	NKv4yOc
43492	NldASpP	JFM1BiF	NKv4yOc
49432	NldASpP	JFM1BiF	NKv4yOc
18737	NldASpP	JFM1BiF	NKv4yOc
26141	NldASpP	JFM1BiF	NKv4yOc
49410	NldASpP	JFM1BiF	NKv4yOc
1520	NldASpP	JFM1BiF	NKv4yOc
47382	NldASpP	JFM1BiF	NKv4yOc
2884	NldASpP	JFM1BiF	NKv4yOc
21746	NldASpP	JFM1BiF	NKv4yOc
6355	NldASpP	JFM1BiF	NKv4yOc
25486	NldASpP	JFM1BiF	NKv4yOc
18954	NldASpP	JFM1BiF	NKv4yOc
2	NldASpP	JFM1BiF	NKv4yOc
15538	NldASpP	JFM1BiF	NKv4yOc
9423	NldASpP	JFM1BiF	NKv4yOc
442	NldASpP	JFM1BiF	NKv4yOc
30698	NldASpP	JFM1BiF	NKv4yOc

	Var198	Var220	Var222
39698	NIdASpP	JFM1BiF	NKv4yOc
16570	NIdASpP	JFM1BiF	NKv4yOc
40080	NIdASpP	JFM1BiF	NKv4yOc
15642	NIdASpP	JFM1BiF	NKv4yOc
40414	NIdASpP	JFM1BiF	NKv4yOc
48929	NIdASpP	JFM1BiF	NKv4yOc
46569	NIdASpP	JFM1BiF	NKv4yOc
36477	NIdASpP	JFM1BiF	NKv4yOc
43960	NIdASpP	JFM1BiF	NKv4yOc
17160	NIdASpP	JFM1BiF	NKv4yOc
17269	NIdASpP	JFM1BiF	NKv4yOc
40771	NIdASpP	JFM1BiF	NKv4yOc
23993	NIdASpP	JFM1BiF	NKv4yOc
16514	NIdASpP	JFM1BiF	NKv4yOc
5184	NIdASpP	JFM1BiF	NKv4yOc
39608	NIdASpP	JFM1BiF	NKv4yOc
1860	NIdASpP	JFM1BiF	NKv4yOc
20568	NIdASpP	JFM1BiF	NKv4yOc
31782	NIdASpP	JFM1BiF	NKv4yOc
25616	NIdASpP	JFM1BiF	NKv4yOc
21395	NIdASpP	JFM1BiF	NKv4yOc
40437	NIdASpP	JFM1BiF	NKv4yOc

In [ ]:

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

Out[ ]:

	Var198	Var220	Var222
43134	ka_ns41	1YVfGrO	fXVEsaq
4461	ka_ns41	1YVfGrO	fXVEsaq
4516	ka_ns41	1YVfGrO	fXVEsaq
16947	ka_ns41	1YVfGrO	fXVEsaq
18081	ka_ns41	1YVfGrO	fXVEsaq
...	...	...	...
32893	ka_ns41	1YVfGrO	fXVEsaq
23716	ka_ns41	1YVfGrO	fXVEsaq
9999	ka_ns41	1YVfGrO	fXVEsaq
43061	ka_ns41	1YVfGrO	fXVEsaq
38	ka_ns41	1YVfGrO	fXVEsaq

93 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)**

- For ex: For column Var198, value 'ka_ns41' always occur with 'FYVIGrO' (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_upselling =  X_test_upselling.drop(['Var220','Var222'], axis = 1)
```

In [ ]:

```
X_test_upselling.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_upselling.pickle', 'wb') as handle:
    pickle.dump(data_new, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [ ]:

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

In [ ]:

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

In [ ]:

```
with open('y_test_upselling.pickle', 'wb') as handle:
    pickle.dump(y_test_upselling, 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)
```