A2Z Insurance

Authors:

- Ianis Rușitoru (20220620)
- Nichita Zamisnii (20220617)
- Ricardo Almeida (20220584)

2022/2023

Index

Import

Back to Index

Import libraries

In [1]:
```python
import warnings
warnings.filterwarnings('ignore')
import os, sys
#hide the warnings

sys.stderr = open(os.devnull, "w")  # silence stderr
from sklearn.ensemble import RandomForestRegressor
sys.stderr = sys.__stderr__  # unsilence stderr


import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from math import ceil
import numpy as np
import pandas as pd
import csv
from sklearn import preprocessing
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, silhouette_samples
import matplotlib.cm as cm
from sompy.visualization.mapview import View2D
from sompy.visualization.bmuhits import BmuHitsView
from sompy.visualization.hitmap import HitMapView
from sklearn.manifold import TSNE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from statsmodels.stats.outliers_influence import variance_inflation_factor
import scipy.stats as stats
from scipy.stats import chi2_contingency
from sklearn.linear_model import LogisticRegression, LinearRegression, LassoCV
from sklearn.feature_selection import RFE, mutual_info_classif, mutual_info_regression
from sklearn.metrics import r2_score, explained_variance_score, mean_absolute_error, mean_squared_error
from sklearn.metrics import median_absolute_error, mutual_info_score
from kmodes.kprototypes import KPrototypes
from sklearn_extra.cluster import KMedoids
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram
from sklearn.cluster import DBSCAN, KMeans, AgglomerativeClustering
from sklearn.base import clone
import hypertools as hyp
from plotnine import *
from sklearn.metrics import pairwise_distances


import umap
import umap.plot
import umap
from sklearn.datasets import fetch_openml
from sklearn.utils import resample


%matplotlib inline
# for better resolution plots
%config InlineBackend.figure_format = 'retina' # optionally, you can change 'svg' to 'retina'

# Seeting seaborn style
sns.set(rc={'figure.figsize':(11.7,8.27)})
```

Import the dataset

In [2]:
```
df = pd.read_sas('a2z_insurance.sas7bdat')
df.head()
```

Out[2]:

	CustID	FirstPolYear	BirthYear	EducDeg	MonthSal	GeoLivArea	Children	CustMonVal	ClaimsRate	PremMotor	PremHousehold	PremHealth	PremLife	Pre
0	1.0	1985.0	1982.0	b'2 - High School'	2177.0	1.0	1.0	380.97	0.39	375.85	79.45	146.36	47.01	
1	2.0	1981.0	1995.0	b'2 - High School'	677.0	4.0	1.0	-131.13	1.12	77.46	416.20	116.69	194.48	
2	3.0	1991.0	1970.0	b'1 - Basic'	2277.0	3.0	0.0	504.67	0.28	206.15	224.50	124.58	86.35	
3	4.0	1990.0	1981.0	b'3 - BSc/MSc'	1099.0	4.0	1.0	-16.99	0.99	182.48	43.35	311.17	35.34	
4	5.0	1986.0	1973.0	b'3 - BSc/MSc'	1763.0	4.0	1.0	35.23	0.90	338.62	47.80	182.59	18.78	

Data Exploration

[Back to Index]

Data

In [3]:
```
df.head()
```

Out[3]:

	CustID	FirstPolYear	BirthYear	EducDeg	MonthSal	GeoLivArea	Children	CustMonVal	ClaimsRate	PremMotor	PremHousehold	PremHealth	PremLife	Pre
0	1.0	1985.0	1982.0	b'2 - High School'	2177.0	1.0	1.0	380.97	0.39	375.85	79.45	146.36	47.01	
1	2.0	1981.0	1995.0	b'2 - High School'	677.0	4.0	1.0	-131.13	1.12	77.46	416.20	116.69	194.48	
2	3.0	1991.0	1970.0	b'1 - Basic'	2277.0	3.0	0.0	504.67	0.28	206.15	224.50	124.58	86.35	
3	4.0	1990.0	1981.0	b'3 - BSc/MSc'	1099.0	4.0	1.0	-16.99	0.99	182.48	43.35	311.17	35.34	
4	5.0	1986.0	1973.0	b'3 - BSc/MSc'	1763.0	4.0	1.0	35.23	0.90	338.62	47.80	182.59	18.78	

In [4]:
```
df.set_index('CustID',inplace=True)
df
```

Out[4]:

	FirstPolYear	BirthYear	EducDeg	MonthSal	GeoLivArea	Children	CustMonVal	ClaimsRate	PremMotor	PremHousehold	PremHealth	PremLife	PremW
CustID													
1.0	1985.0	1982.0	b'2 - High School'	2177.0	1.0	1.0	380.97	0.39	375.85	79.45	146.36	47.01	1
2.0	1981.0	1995.0	b'2 - High School'	677.0	4.0	1.0	-131.13	1.12	77.46	416.20	116.69	194.48	10
3.0	1991.0	1970.0	b'1 - Basic'	2277.0	3.0	0.0	504.67	0.28	206.15	224.50	124.58	86.35	9
4.0	1990.0	1981.0	b'3 - BSc/MSc'	1099.0	4.0	1.0	-16.99	0.99	182.48	43.35	311.17	35.34	2
5.0	1986.0	1973.0	b'3 - BSc/MSc'	1763.0	4.0	1.0	35.23	0.90	338.62	47.80	182.59	18.78	4
...	...	...	...	...	...	...	...	...	...	...	...	...	
10292.0	1984.0	1949.0	b'4 - PhD'	3188.0	2.0	0.0	-0.11	0.96	393.74	49.45	173.81	9.78	1
10293.0	1977.0	1952.0	b'1 - Basic'	2431.0	3.0	0.0	1405.60	0.00	133.58	1035.75	143.25	12.89	10
10294.0	1994.0	1976.0	b'3 - BSc/MSc'	2918.0	1.0	1.0	524.10	0.21	403.63	132.80	142.25	12.67	
10295.0	1981.0	1977.0	b'1 - Basic'	1971.0	2.0	1.0	250.05	0.65	188.59	211.15	198.37	63.90	11
10296.0	1990.0	1981.0	b'4 - PhD'	2815.0	1.0	1.0	463.75	0.27	414.08	94.45	141.25	6.89	1

10296 rows × 13 columns

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Float64Index: 10296 entries, 1.0 to 10296.0
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   FirstPolYear   10266 non-null  float64
 1   BirthYear      10279 non-null  float64
 2   EducDeg        10279 non-null  object 
 3   MonthSal       10260 non-null  float64
 4   GeoLivArea     10295 non-null  float64
 5   Children       10275 non-null  float64
 6   CustMonVal     10296 non-null  float64
 7   ClaimsRate     10296 non-null  float64
 8   PremMotor      10262 non-null  float64
 9   PremHousehold  10296 non-null  float64
 10  PremHealth     10253 non-null  float64
 11  PremLife       10192 non-null  float64
 12  PremWork       10210 non-null  float64
dtypes: float64(12), object(1)
memory usage: 1.1+ MB
```

In [6]:
```
df.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
FirstPolYear	10266.0	1991.062634	511.267913	1974.00	1980.00	1986.00	1992.0000	53784.00
BirthYear	10279.0	1968.007783	19.709476	1028.00	1953.00	1968.00	1983.0000	2001.00
MonthSal	10260.0	2506.667057	1157.449634	333.00	1706.00	2501.50	3290.2500	55215.00
GeoLivArea	10295.0	2.709859	1.266291	1.00	1.00	3.00	4.0000	4.00
Children	10275.0	0.706764	0.455268	0.00	0.00	1.00	1.0000	1.00
CustMonVal	10296.0	177.892605	1945.811505	-165680.42	-9.44	186.87	399.7775	11875.89
ClaimsRate	10296.0	0.742772	2.916964	0.00	0.39	0.72	0.9800	256.20
PremMotor	10262.0	300.470252	211.914997	-4.11	190.59	298.61	408.3000	11604.42
PremHousehold	10296.0	210.431192	352.595984	-75.00	49.45	132.80	290.0500	25048.80
PremHealth	10253.0	171.580833	296.405976	-2.11	111.80	162.81	219.8200	28272.00
PremLife	10192.0	41.855782	47.480632	-7.00	9.89	25.56	57.7900	398.30
PremWork	10210.0	41.277514	51.513572	-12.00	10.67	25.67	56.7900	1988.70

In [7]:
```
#change the variables type

df.index = df.index.map(int)
df['FirstPolYear'] = df['FirstPolYear'].astype(pd.Int32Dtype())
df['BirthYear'] = df['BirthYear'].astype(pd.Int32Dtype())
df['GeoLivArea'] = df['GeoLivArea'].astype(pd.Int32Dtype())
df['Children'] = df['Children'].astype(pd.Int32Dtype())
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10296 entries, 1 to 10296
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   FirstPolYear   10266 non-null  Int32  
 1   BirthYear      10279 non-null  Int32  
 2   EducDeg        10279 non-null  object 
 3   MonthSal       10260 non-null  float64
 4   GeoLivArea     10295 non-null  Int32  
 5   Children       10275 non-null  Int32  
 6   CustMonVal     10296 non-null  float64
 7   ClaimsRate     10296 non-null  float64
 8   PremMotor      10262 non-null  float64
 9   PremHousehold  10296 non-null  float64
 10  PremHealth     10253 non-null  float64
 11  PremLife       10192 non-null  float64
 12  PremWork       10210 non-null  float64
dtypes: Int32(4), float64(8), object(1)
memory usage: 1005.5+ KB
```

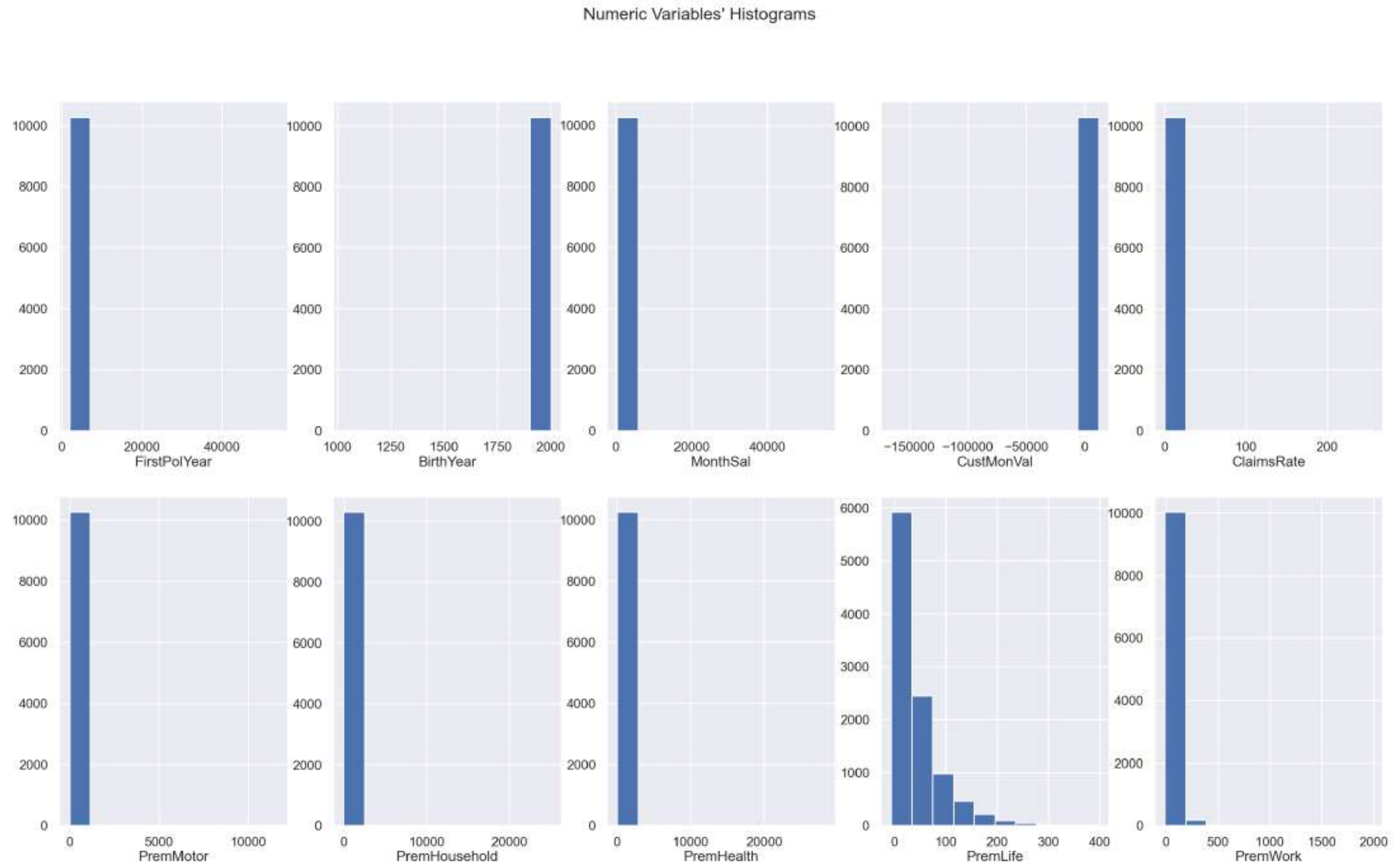
Visualization

Back to Index

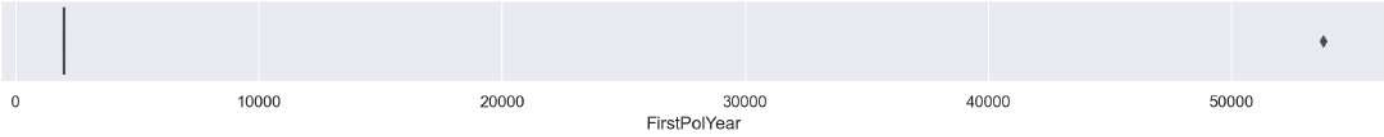
In [9]:
```python
#define metric and non-metric

non_metric_features = ["Children","EducDeg",'GeoLivArea']
metric_features = df.columns.drop(non_metric_features).to_list()
metric_features
```

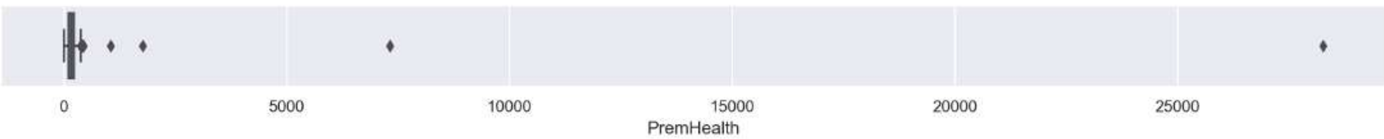
Out[9]:
```
['FirstPolYear',
 'BirthYear',
 'MonthSal',
 'CustMonVal',
 'ClaimsRate',
 'PremMotor',
 'PremHousehold',
 'PremHealth',
 'PremLife',
 'PremWork']
```

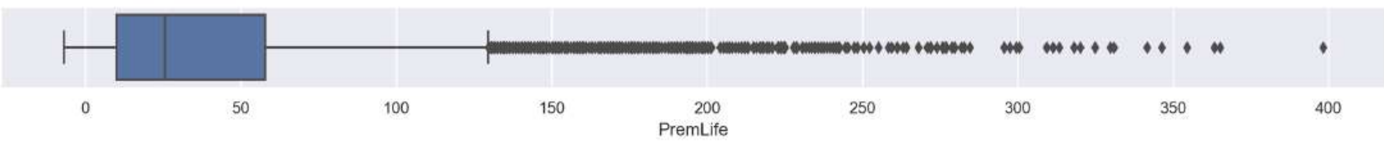
In [10]:
```python
# All Numeric Variables' Histograms in one figure
sns.set()

# Prepare figure. Create individual axes where each histogram will be placed
fig, axes = plt.subplots(2, ceil(len(metric_features) / 2), figsize=(20, 11))

# Plot data
# Iterate across axes objects and associate each histogram (hint: use the ax.hist() instead of plt.hist()):
for ax, feat in zip(axes.flatten(), metric_features): # Notice the zip() function and flatten() method
    ax.hist(df[feat][~np.isnan(df[feat])])
    ax.set_title(feat, y=-0.13)

# Layout
# Add a centered title to the figure:
title = "Numeric Variables' Histograms"

plt.suptitle(title)

plt.show()
```



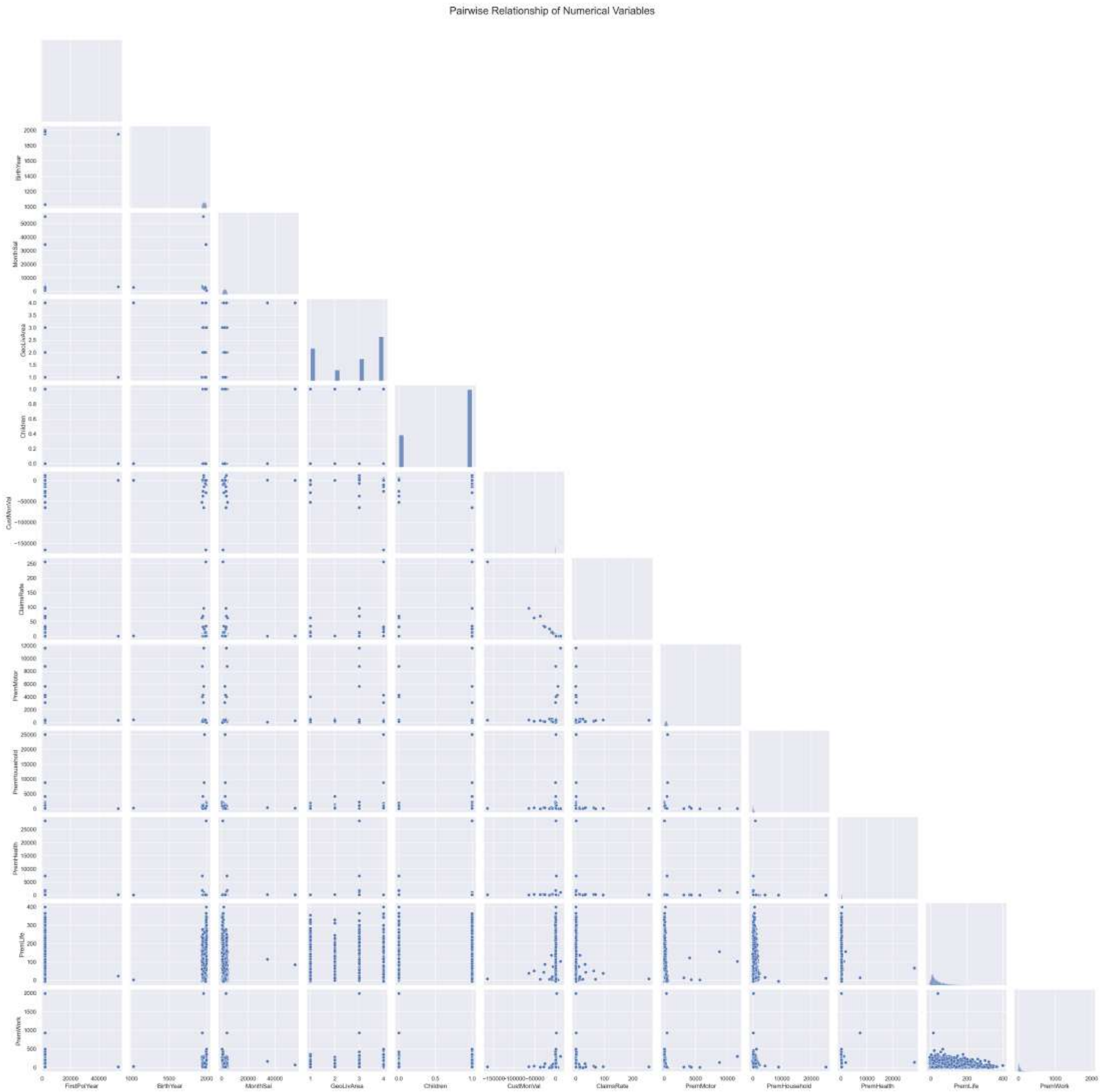
In [11]:
```
for column in df[metric_features]:
        plt.figure(figsize=(17,1))
        sns.boxplot(data=df, x=column)
```







In [12]:
```
sns.set()
# Setting pairplot
sns.pairplot(df, diag_kind="hist",corner=True)
# Layout
plt.subplots_adjust(top=0.95)
plt.suptitle("Pairwise Relationship of Numerical Variables", fontsize=20)

plt.show()
```

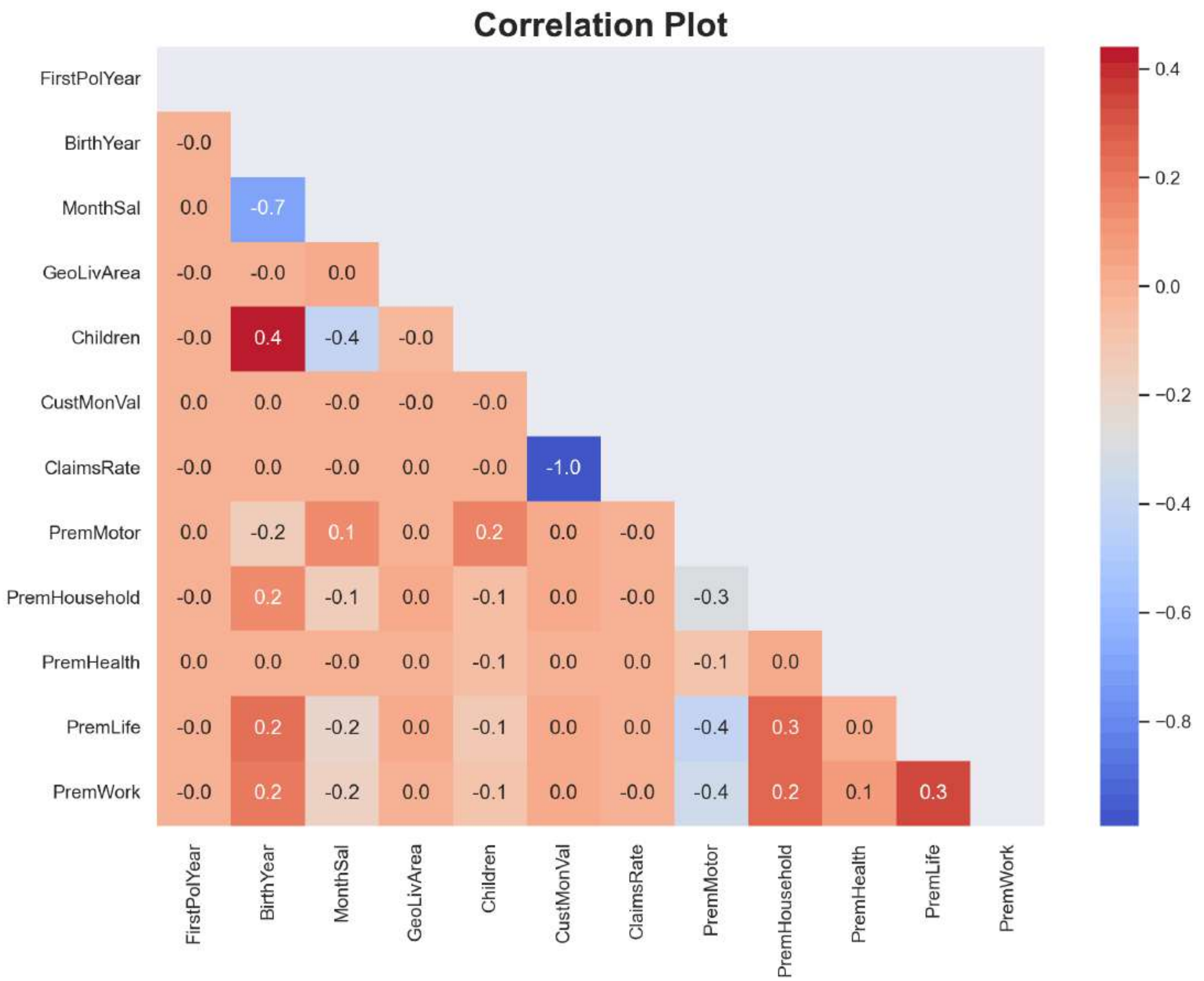


Correlation Check

Back to Index

In [13]:
```
# plot the heatmap
def corr_heatmap(corr):
    sns.heatmap(data = corr, annot = True, cmap = sns.color_palette("coolwarm", 50),
                fmt = '0.1f', mask = np.triu(corr))
    plt.title("Correlation Plot", fontsize = 20, fontweight = 'bold')
    plt.show()
```

In [14]:
```
df_corr = df.corr()
corr_heatmap(df_corr)
```



In [15]:
```
df_corr
```

Out[15]:

	FirstPolYear	BirthYear	MonthSal	GeoLivArea	Children	CustMonVal	ClaimsRate	PremMotor	PremHousehold	PremHealth	PremLife	PremW
FirstPolYear	1.000000	-0.010300	0.006808	-0.013397	-0.015502	0.000821	-0.001209	0.002314	-0.005597	0.000576	-0.003924	-0.004
BirthYear	-0.010300	1.000000	-0.696189	-0.016567	0.439691	0.003498	0.004473	-0.157014	0.150434	0.003248	0.233701	0.208
MonthSal	0.006808	-0.696189	1.000000	0.015385	-0.393768	-0.003256	-0.003510	0.135842	-0.133248	-0.002123	-0.196412	-0.174
GeoLivArea	-0.013397	-0.016567	0.015385	1.000000	-0.021602	-0.005789	0.007498	0.000737	0.011035	0.003250	0.011742	0.003
Children	-0.015502	0.439691	-0.393768	-0.021602	1.000000	-0.000928	-0.002202	0.155954	-0.063026	-0.065399	-0.116160	-0.085
CustMonVal	0.000821	0.003498	-0.003256	-0.005789	-0.000928	1.000000	-0.992622	0.033655	0.032664	0.000953	0.010432	0.020
ClaimsRate	-0.001209	0.004473	-0.003510	0.007498	-0.002202	-0.992622	1.000000	-0.006459	-0.007958	0.006083	0.001081	-0.001
PremMotor	0.002314	-0.157014	0.135842	0.000737	0.155954	0.033655	-0.006459	1.000000	-0.276720	-0.077721	-0.409424	-0.350
PremHousehold	-0.005597	0.150434	-0.133248	0.011035	-0.063026	0.032664	-0.007958	-0.276720	1.000000	0.024844	0.261867	0.240
PremHealth	0.000576	0.003248	-0.002123	0.003250	-0.065399	0.000953	0.006083	-0.077721	0.024844	1.000000	0.027125	0.080
PremLife	-0.003924	0.233701	-0.196412	0.011742	-0.116160	0.010432	0.001081	-0.409424	0.261867	0.027125	1.000000	0.344
PremWork	-0.004296	0.208049	-0.174798	0.003561	-0.085566	0.020290	-0.001468	-0.350368	0.240004	0.080007	0.344420	1.000

Preprocessing

Duplicates

[Back to Index](#)

In [16]:
```
df[df.duplicated()]
```

Out[16]:

CustID	FirstPolYear	BirthYear	EducDeg	MonthSal	GeoLivArea	Children	CustMonVal	ClaimsRate	PremMotor	PremHousehold	PremHealth	PremLife	PremW
8014	1987	1987	b'2 - High School'	1912.0	4	1	290.61	0.58	202.37	177.25	306.39	63.90	-(
8122	1977	1974	b'2 - High School'	2204.0	4	1	-22.11	1.00	214.93	88.90	266.94	39.23	42
9554	1986	1952	b'2 - High School'	3900.0	4	0	-119.35	1.10	163.03	481.75	224.82	94.35	18

In [17]:
```
len(df)
```

Out[17]: 10296

In [18]:
```
df.drop_duplicates(inplace = True)
```

In [19]:
```
len(df)
```

Out[19]: 10293

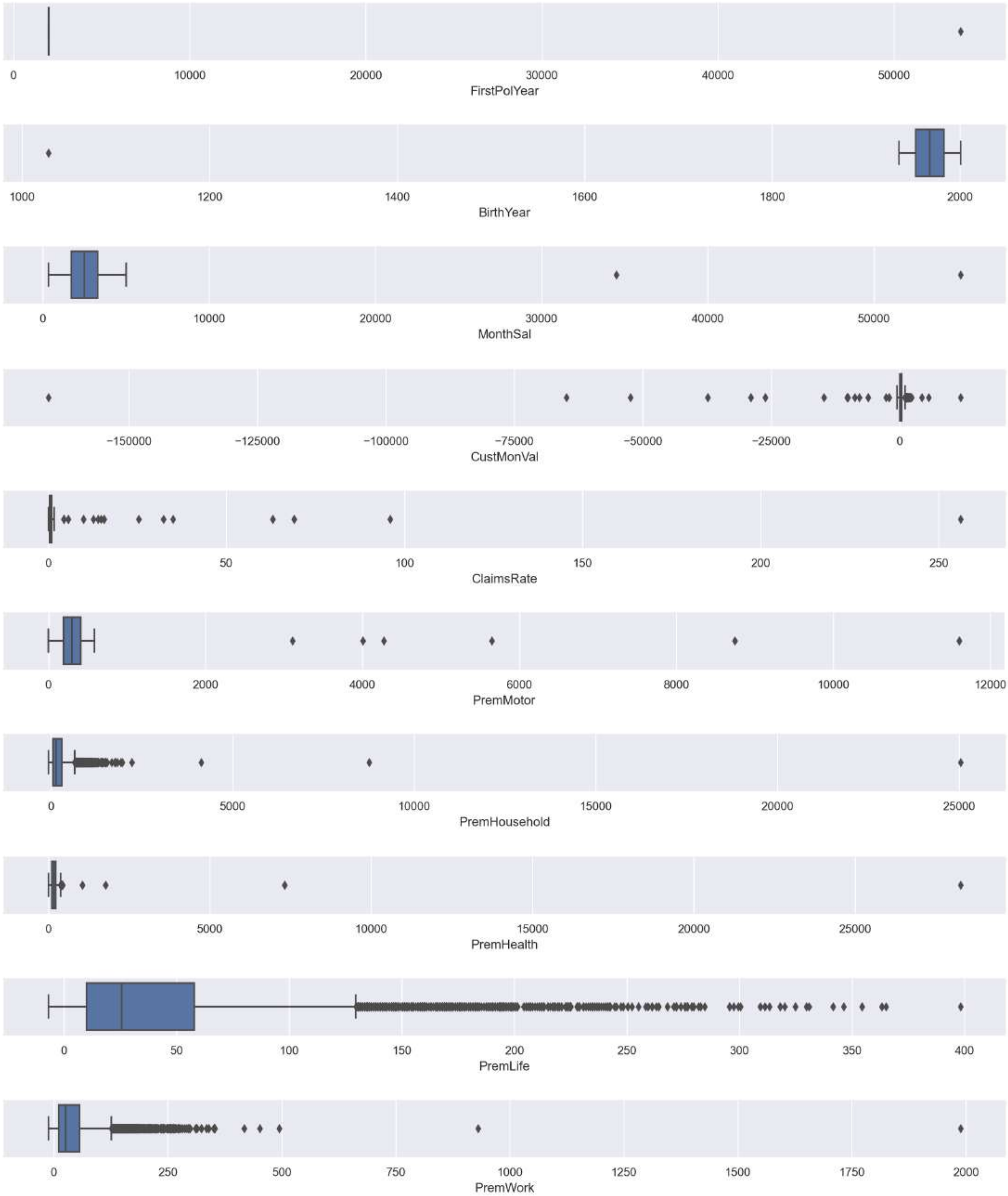
Outliers Removal

Back to Index

In [20]:
```
non_metric_features = ["Children","EducDeg",'GeoLivArea']
metric_features = df.columns.drop(non_metric_features).to_list()
metric_features
```

Out[20]:
```
['FirstPolYear',
 'BirthYear',
 'MonthSal',
 'CustMonVal',
 'ClaimsRate',
 'PremMotor',
 'PremHousehold',
 'PremHealth',
 'PremLife',
 'PremWork']
```

```
In [21]: for column in df[metric_features]:
             plt.figure(figsize=(17,1))
             sns.boxplot(data=df, x=column)
```



In [22]:
```python
q25 = df.quantile(.25)
q75 = df.quantile(.75)
iqr = (q75 - q25)

upper_lim = q75 + 1.5 * iqr
lower_lim = q25 - 1.5 * iqr

for column in df[metric_features]:
    llim = lower_lim[column]
    ulim = upper_lim[column]
    print(column,": ",llim,",",ulim)
```

```
FirstPolYear :  1962.0 , 2010.0
BirthYear :  1908.0 , 2028.0
MonthSal :  -670.0 , 5666.0
CustMonVal :  -623.3900000000001 , 1013.8100000000001
ClaimsRate :  -0.495 , 1.865
PremMotor :  -135.975 , 734.865
PremHousehold :  -311.45000000000005 , 650.95
PremHealth :  -49.05999999999999 , 379.9
PremLife :  -61.959999999999994 , 129.64
PremWork :  -58.50999999999999 , 125.97
```

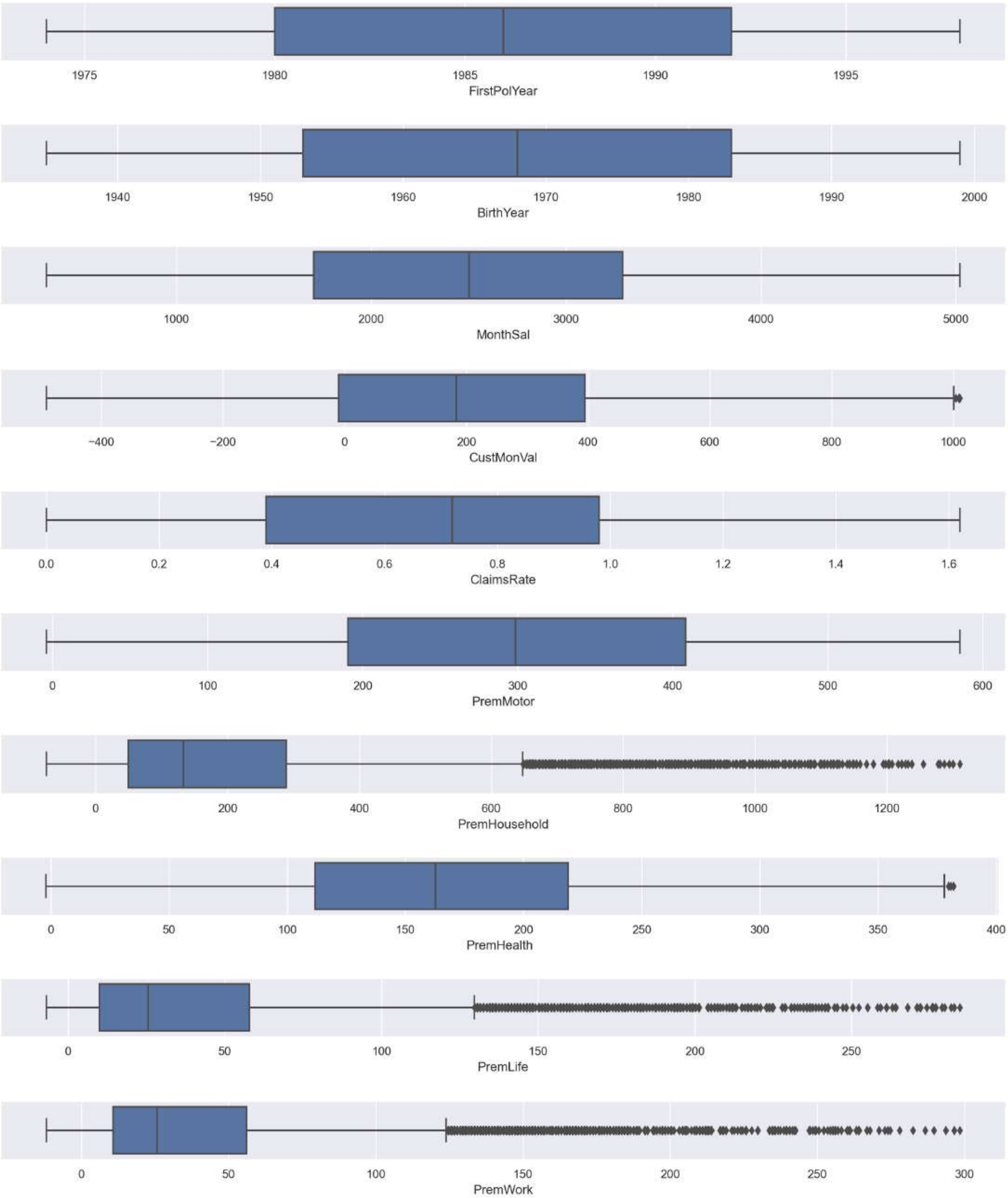
In [23]:
```python
df_no_outliers = df.copy()
```

In [24]:
```python
filters = (
    (df_no_outliers['FirstPolYear'] <= 2010) &
    (df_no_outliers['BirthYear'] <= 2000) & (df_no_outliers['BirthYear'] >= 1897) &
    (df_no_outliers['CustMonVal'] <= 1013) & (df_no_outliers['CustMonVal'] >= -624) &
    (df_no_outliers['MonthSal'] <= 5666) &
    (df_no_outliers['ClaimsRate'] >= -0.495) & (df_no_outliers['ClaimsRate'] <= 1.865) &
    (df_no_outliers['PremMotor'] <= 800 ) &
    (df_no_outliers['PremHousehold'] < 1350) &
    (df_no_outliers['PremHealth'] < 382) &
    (df_no_outliers['PremLife'] < 285) &
    (df_no_outliers['PremWork'] < 300)

)

print('Percentage of data kept after removing outliers:',
      np.round(df_no_outliers[filters].shape[0] / df_no_outliers.shape[0], 4))

#df_no_outliers = df_no_outliers[filters]
```

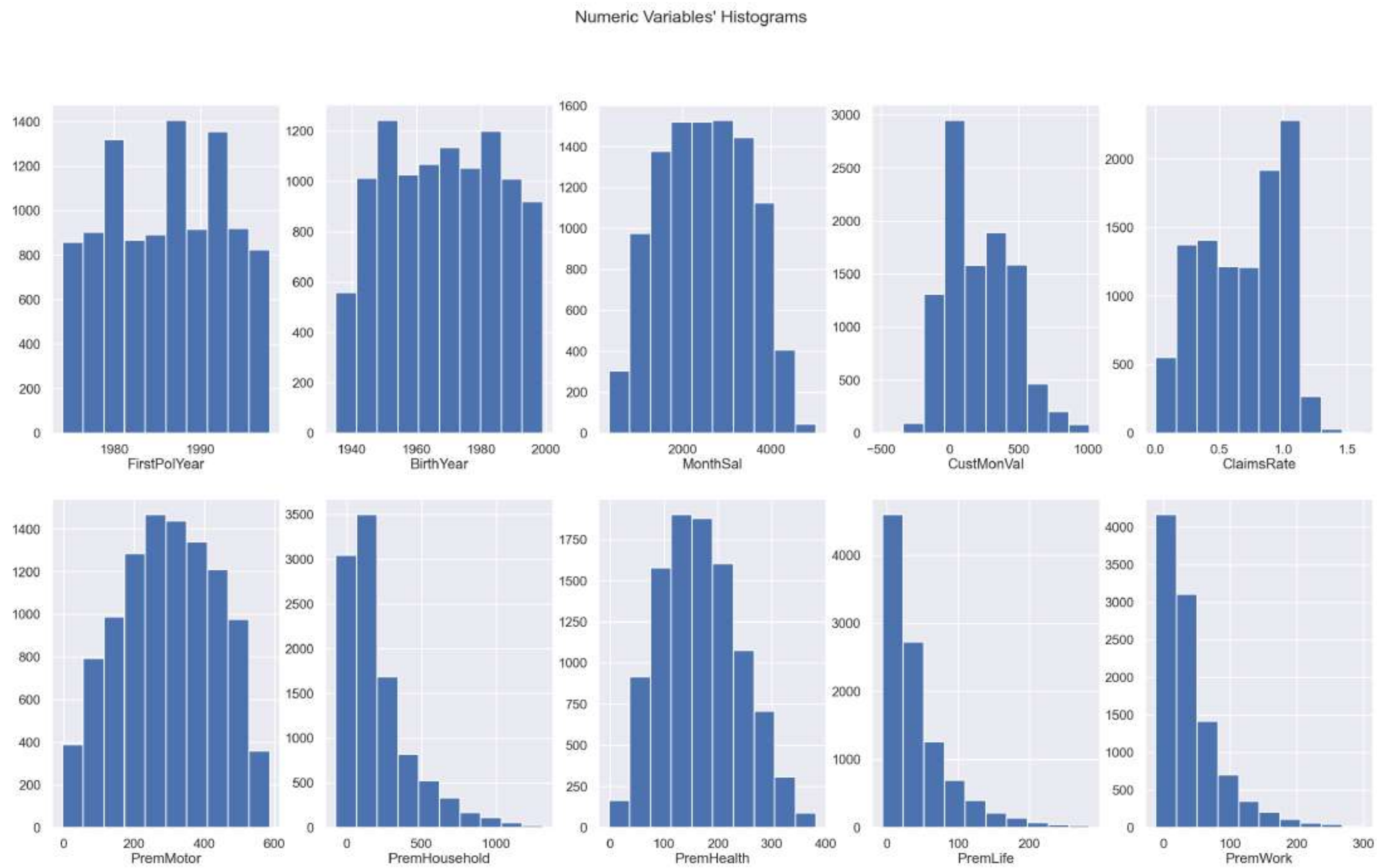
```
Percentage of data kept after removing outliers: 0.9525
```

In [25]:
```python
df_no_outliers['FirstPolYear'][(df_no_outliers['FirstPolYear'] >= 2010)] = pd.NA
df_no_outliers['BirthYear'][(df_no_outliers['BirthYear'] >= 2000) | (df_no_outliers['BirthYear'] <= 1897)] = pd.NA
df_no_outliers['CustMonVal'][(df_no_outliers['CustMonVal'] <= -624) | (df_no_outliers['CustMonVal'] >= 1013)] = pd.NA
df_no_outliers['ClaimsRate'][(df_no_outliers['ClaimsRate'] <= -0.495) | (df_no_outliers['ClaimsRate'] >= 1.865)] = pd.NA
df_no_outliers['MonthSal'][(df_no_outliers['MonthSal'] >= 5666)] = pd.NA
df_no_outliers['PremMotor'][(df_no_outliers['PremMotor'] >= 800)] = pd.NA
df_no_outliers['PremHousehold'][(df_no_outliers['PremHousehold'] >= 1350)] = pd.NA
df_no_outliers['PremHealth'][(df_no_outliers['PremHealth'] >= 382)] = pd.NA
df_no_outliers['PremLife'][(df_no_outliers['PremLife'] >= 285)] = pd.NA
df_no_outliers['PremWork'][(df_no_outliers['PremWork'] >= 300)] = pd.NA
```

In [26]:
```
for column in df_no_outliers[metric_features]:
        plt.figure(figsize=(17,1))
        sns.boxplot(data=df_no_outliers, x=column)
```



In [27]:
```
# All Numeric Variables' Histograms in one figure
sns.set()

# Prepare figure. Create individual axes where each histogram will be placed
fig, axes = plt.subplots(2, ceil(len(metric_features) / 2), figsize=(20, 11))

# Plot data
# Iterate across axes objects and associate each histogram (hint: use the ax.hist() instead of plt.hist()):
for ax, feat in zip(axes.flatten(), metric_features): # Notice the zip() function and flatten() method
    ax.hist(df_no_outliers[feat][~np.isnan(df_no_outliers[feat])])
    ax.set_title(feat, y=-0.13)

# Layout
# Add a centered title to the figure:
title = "Numeric Variables' Histograms"

plt.suptitle(title)

plt.show()
```



Missing Values

Back to Index

Non-Metric Features

In [28]:
```
for column in df_no_outliers[non_metric_features]:
    print(df_no_outliers[column].value_counts())
```

```
1    7260
0    3012
Name: Children, dtype: Int64
b'3 - BSc/MSc'        4799
b'2 - High School'    3507
b'1 - Basic'          1272
b'4 - PhD'             698
Name: EducDeg, dtype: int64
4    4142
1    3048
3    2066
2    1036
Name: GeoLivArea, dtype: Int64
```

In [29]:
```
df_no_outliers[non_metric_features].isna().sum()
```

Out[29]:
```
Children      21
EducDeg       17
GeoLivArea     1
dtype: int64
```

Education

In [30]:
```
df_no_outliers['EducDeg'].mode()
```

Out[30]:
```
0    b'3 - BSc/MSc'
Name: EducDeg, dtype: object
```

In [31]:
```
df_no_outliers['EducDeg'].fillna(df_no_outliers['EducDeg'].mode()[0], inplace=True)
```

In [32]:
```
df_no_outliers[non_metric_features].isna().sum()
```

Out[32]:
```
Children      21
EducDeg        0
GeoLivArea     1
dtype: int64
```

GeoLivArea

In [33]:
```
df_no_outliers['GeoLivArea'].mode()
```

Out[33]:
```
0    4
Name: GeoLivArea, dtype: Int32
```

In [34]:
```
df_no_outliers['GeoLivArea'].fillna(df_no_outliers['GeoLivArea'].mode()[0], inplace=True)
```

In [35]:
```
df_no_outliers[non_metric_features].isna().sum()
```

Out[35]:
```
Children      21
EducDeg        0
GeoLivArea     0
dtype: int64
```

Children

In [36]:
```
df_no_outliers['Children'].mode()
```

Out[36]:
```
0    1
Name: Children, dtype: Int32
```

In [37]:
```
df_no_outliers['Children'].fillna(df_no_outliers['Children'].mode()[0], inplace=True)
```

In [38]:
```
df_no_outliers[non_metric_features].isna().sum()
```

Out[38]:
```
Children      0
EducDeg       0
GeoLivArea    0
dtype: int64
```

Metric Features

In [39]:
```
metric_features
```

Out[39]:
```
['FirstPolYear',
 'BirthYear',
 'MonthSal',
 'CustMonVal',
 'ClaimsRate',
 'PremMotor',
 'PremHousehold',
 'PremHealth',
 'PremLife',
 'PremWork']
```

In [40]:
```
df_no_outliers[metric_features].isna().sum()
```

Out[40]:
```
FirstPolYear      31
BirthYear         65
MonthSal          38
CustMonVal       110
ClaimsRate        15
PremMotor         40
PremHousehold     30
PremHealth        65
PremLife         122
PremWork         103
dtype: int64
```

FirstPolYear

In [41]:
```
df_no_outliers['FirstPolYear'].median()
```

Out[41]:
```
1986.0
```

In [42]:
```
df_no_outliers['FirstPolYear'].fillna(df_no_outliers['FirstPolYear'].median(), inplace=True)
```

In [43]:
```
df_no_outliers[metric_features].isna().sum()
```

Out[43]:
```
FirstPolYear       0
BirthYear         65
MonthSal          38
CustMonVal       110
ClaimsRate        15
PremMotor         40
PremHousehold     30
PremHealth        65
PremLife         122
PremWork         103
dtype: int64
```

Age

In [44]:
```
df_no_outliers['BirthYear'].median()
```

Out[44]:
```
1968.0
```

In [45]:
```
df_no_outliers['BirthYear'].fillna(df_no_outliers['BirthYear'].median(), inplace=True)
```

In [46]:
```
df_no_outliers[metric_features].isna().sum()
```

Out[46]:
```
FirstPolYear       0
BirthYear          0
MonthSal          38
CustMonVal       110
ClaimsRate        15
PremMotor         40
PremHousehold     30
PremHealth        65
PremLife         122
PremWork         103
dtype: int64
```

Monthly Salary

In [47]:
```
df_no_outliers['MonthSal'].median()
```

Out[47]:
```
2501.0
```

In [48]:
```
df_no_outliers['MonthSal'].fillna(df_no_outliers['MonthSal'].median(), inplace=True)
#mean or median(); No need of [0] because it returns a float value.
```

In [49]:
```
df_no_outliers[metric_features].isna().sum()
```

Out[49]:
```
FirstPolYear       0
BirthYear          0
MonthSal           0
CustMonVal       110
ClaimsRate        15
PremMotor         40
PremHousehold     30
PremHealth        65
PremLife         122
PremWork         103
dtype: int64
```

CustMonVal

In [50]:
```python
df_no_outliers['CustMonVal'].median()
```

Out[50]:
```
183.82
```

In [51]:
```python
df_no_outliers['CustMonVal'].fillna(df_no_outliers['CustMonVal'].median(), inplace=True)
#mean or median(); No need of [0] because it returns a float value.
```

In [52]:
```python
df_no_outliers[metric_features].isna().sum()
```

Out[52]:
```
FirstPolYear       0
BirthYear          0
MonthSal           0
CustMonVal         0
ClaimsRate        15
PremMotor         40
PremHousehold     30
PremHealth        65
PremLife         122
PremWork         103
dtype: int64
```

ClaimsRate

In [53]:
```python
df_no_outliers['ClaimsRate'].median()
```

Out[53]:
```
0.72
```

In [54]:
```python
df_no_outliers['ClaimsRate'].fillna(df_no_outliers['ClaimsRate'].median(), inplace=True)
#mean or median(); No need of [0] because it returns a float value.
```

In [55]:
```python
df_no_outliers[metric_features].isna().sum()
```

Out[55]:
```
FirstPolYear       0
BirthYear          0
MonthSal           0
CustMonVal         0
ClaimsRate         0
PremMotor         40
PremHousehold     30
PremHealth        65
PremLife         122
PremWork         103
dtype: int64
```

Premiums

In [56]:
```
df_no_outliers.loc[df_no_outliers['PremHousehold']==0]
```

Out[56]:

CustID	FirstPolYear	BirthYear	EducDeg	MonthSal	GeoLivArea	Children	CustMonVal	ClaimsRate	PremMotor	PremHousehold	PremHealth	PremLife	PremW
489	1977	1947	b'3 - BSc/MSc'	2501.0	3	0	-52.56	1.09	NaN	0.0	278.83	NaN	27
540	1979	1992	b'2 - High School'	2501.0	2	1	85.13	0.79	156.25	0.0	237.71	12.89	110
630	1987	1968	b'3 - BSc/MSc'	2483.0	3	1	-37.00	1.02	523.32	0.0	69.79	-5.00	-7
831	1988	1942	b'2 - High School'	2501.0	4	0	475.43	0.00	274.83	0.0	180.59	25.45	19
863	1987	1981	b'3 - BSc/MSc'	2127.0	1	1	-25.00	0.00	NaN	0.0	NaN	NaN	N
913	1998	1980	b'2 - High School'	2093.0	4	1	295.95	0.46	296.39	0.0	271.05	17.67	6
1060	1977	1967	b'3 - BSc/MSc'	3297.0	1	1	484.43	0.12	511.43	0.0	58.79	6.89	4
1134	1974	1952	b'3 - BSc/MSc'	3560.0	2	0	-25.00	0.00	NaN	0.0	NaN	NaN	N
1161	1981	1983	b'3 - BSc/MSc'	2501.0	3	1	4.56	0.79	NaN	0.0	106.02	3.89	33
1290	1980	1944	b'3 - BSc/MSc'	3962.0	4	0	-123.46	1.16	184.70	0.0	336.84	45.23	53
1377	1983	1977	b'3 - BSc/MSc'	1460.0	3	1	184.15	0.62	285.83	0.0	182.70	76.68	N
1781	1998	1964	b'3 - BSc/MSc'	2501.0	1	1	454.43	0.16	509.43	0.0	66.90	1.89	-10
1817	1978	1959	b'3 - BSc/MSc'	2501.0	2	1	58.68	0.62	NaN	0.0	138.47	17.78	65
2227	1976	1952	b'2 - High School'	2825.0	4	1	253.83	0.52	278.83	0.0	121.58	124.69	56
2565	1992	1941	b'3 - BSc/MSc'	4843.0	4	0	-48.45	1.04	231.49	0.0	306.28	11.78	23
2594	1994	1948	b'2 - High School'	3745.0	4	0	391.64	0.26	182.59	0.0	146.14	234.49	-0
2722	1996	1971	b'3 - BSc/MSc'	2596.0	4	1	375.41	0.32	384.85	0.0	163.03	24.56	15
2892	1985	1982	b'3 - BSc/MSc'	1343.0	1	1	-63.67	1.07	433.97	0.0	145.36	20.67	-6
3130	1982	1968	b'3 - BSc/MSc'	2827.0	4	1	487.43	0.12	496.65	0.0	63.68	15.78	5
3166	1995	1981	b'2 - High School'	1813.0	4	1	-25.00	0.00	NaN	0.0	NaN	NaN	N
3228	1990	1949	b'4 - PhD'	2501.0	1	0	8.12	0.87	225.60	0.0	NaN	19.56	13
3318	1980	1943	b'2 - High School'	2501.0	2	0	-38.56	1.02	363.29	0.0	154.14	47.23	13
3451	1977	1966	b'3 - BSc/MSc'	3094.0	4	0	523.99	0.08	541.88	0.0	46.23	0.89	8
3973	1988	1979	b'2 - High School'	2501.0	4	1	207.60	0.51	252.27	0.0	134.58	19.67	72
4023	1995	1988	b'3 - BSc/MSc'	1296.0	3	1	-25.00	0.00	NaN	0.0	NaN	NaN	N
4114	1991	1993	b'2 - High School'	1073.0	1	1	-25.00	0.00	NaN	0.0	NaN	NaN	N
4272	1983	1988	b'3 - BSc/MSc'	1458.0	1	1	-25.00	0.00	NaN	0.0	NaN	NaN	N
4626	1978	1981	b'3 - BSc/MSc'	2501.0	4	1	414.86	0.24	449.64	0.0	117.80	9.78	1
4862	1987	1955	b'3 - BSc/MSc'	2740.0	1	1	-23.00	1.00	319.17	0.0	242.49	13.67	15
4909	1974	1982	b'4 - PhD'	1826.0	3	1	401.86	0.29	435.86	0.0	149.36	10.89	1
5050	1984	1971	b'2 - High School'	2923.0	4	1	527.77	0.06	557.77	0.0	29.34	-4.00	7
5083	1986	1953	b'2 - High School'	2501.0	4	1	421.97	0.00	398.74	0.0	NaN	1.89	46
5397	1977	1982	b'3 - BSc/MSc'	2501.0	4	1	4.23	0.95	203.37	0.0	285.61	54.90	25
5400	1983	1969	b'2 - High School'	2697.0	2	1	-31.55	1.01	488.76	0.0	79.46	-0.11	-2
5539	1983	1984	b'3 - BSc/MSc'	2501.0	3	1	323.18	0.35	301.28	0.0	188.37	29.45	17
5847	1985	1936	b'2 - High School'	3812.0	1	0	273.50	0.45	237.71	0.0	222.71	60.79	18
5984	1986	1959	b'3 - BSc/MSc'	2745.0	3	0	-25.00	0.00	NaN	0.0	NaN	NaN	N
6401	1982	1962	b'3 - BSc/MSc'	2643.0	3	1	-25.00	1.00	545.88	0.0	47.23	2.00	3

CustID	FirstPolYear	BirthYear	EducDeg	MonthSal	GeoLivArea	Children	CustMonVal	ClaimsRate	PremMotor	PremHousehold	PremHealth	PremLife	PremW
6440	1991	1956	b'3 - BSc/MSc'	2375.0	2	1	-25.00	0.00	NaN	0.0	NaN	NaN	N
6462	1976	1984	b'3 - BSc/MSc'	2501.0	4	1	-25.22	1.00	389.74	0.0	190.48	-0.11	11
6561	1976	1992	b'3 - BSc/MSc'	2501.0	4	1	85.91	0.77	91.24	0.0	227.60	60.01	110
6615	1995	1997	b'1 - Basic'	1231.0	1	1	-25.00	0.00	NaN	0.0	NaN	NaN	N
6729	1980	1960	b'3 - BSc/MSc'	2701.0	1	0	483.43	0.13	490.65	0.0	73.57	17.78	N
7591	1978	1972	b'3 - BSc/MSc'	2198.0	4	1	-51.89	1.05	557.66	0.0	31.45	8.89	-9
7893	1986	1991	b'3 - BSc/MSc'	2501.0	2	1	-25.00	1.00	335.62	0.0	220.93	8.89	N
8145	1985	1974	b'2 - High School'	1507.0	4	1	505.88	0.08	550.88	0.0	34.23	-5.00	-5
8535	1976	1943	b'3 - BSc/MSc'	3607.0	3	0	-16.22	0.98	327.73	0.0	163.81	29.34	38
8586	1997	1950	b'3 - BSc/MSc'	3269.0	1	0	-25.00	0.00	NaN	0.0	NaN	NaN	N
8659	1990	1971	b'3 - BSc/MSc'	2088.0	4	1	-17.33	0.99	503.76	0.0	75.57	-7.00	28
8677	1979	1997	b'1 - Basic'	2501.0	4	1	-291.16	1.62	65.90	0.0	97.13	130.47	135
8678	1988	1944	b'1 - Basic'	2501.0	1	0	100.58	0.75	125.58	0.0	218.04	101.02	60
8854	1975	1962	b'2 - High School'	3107.0	4	1	36.79	0.87	149.03	0.0	210.93	87.24	29
8918	1978	1995	b'2 - High School'	936.0	3	1	-110.24	1.18	32.56	0.0	267.94	117.80	45
9277	1983	1947	b'3 - BSc/MSc'	2501.0	4	0	41.90	0.89	217.93	0.0	257.27	41.34	66
9399	1974	1956	b'1 - Basic'	2170.0	3	0	-25.00	0.00	NaN	0.0	NaN	NaN	N
9480	1983	1962	b'2 - High School'	3240.0	4	1	-18.00	0.99	555.55	0.0	24.45	7.00	9
9689	1990	1972	b'4 - PhD'	1563.0	4	0	520.88	0.08	553.77	0.0	33.34	6.89	-1
9789	1976	1937	b'3 - BSc/MSc'	4050.0	1	0	-25.00	0.00	NaN	0.0	NaN	NaN	N
9822	1978	1988	b'3 - BSc/MSc'	2501.0	3	1	289.39	0.47	339.84	0.0	175.81	51.12	25
9961	1994	1989	b'2 - High School'	2501.0	1	1	227.38	0.56	171.92	0.0	233.60	80.46	89

In [57]:
```
df_no_outliers.loc[863]
```

Out[57]:
```
FirstPolYear                 1987
BirthYear                    1981
EducDeg             b'3 - BSc/MSc'
MonthSal                   2127.0
GeoLivArea                      1
Children                        1
CustMonVal                  -25.0
ClaimsRate                    0.0
PremMotor                     NaN
PremHousehold                 0.0
PremHealth                    NaN
PremLife                      NaN
PremWork                      NaN
Name: 863, dtype: object
```

In [58]:
```
df_no_outliers.loc[(df_no_outliers['CustMonVal']==-25) & (df_no_outliers['ClaimsRate']==0)]
#CustMonVal = (annual profit from the customer) X (number of years that they are a customer) - (acquisition cost)
#acquisition cost = 25, because ClaimsRate = 0 (Amount paid by the insurance company (€)(0)/ Premiums (€)(0)) - not our client in
```

Out[58]:

CustID	FirstPolYear	BirthYear	EducDeg	MonthSal	GeoLivArea	Children	CustMonVal	ClaimsRate	PremMotor	PremHousehold	PremHealth	PremLife	PremW
863	1987	1981	b'3 - BSc/MSc'	2127.0	1	1	-25.0	0.0	NaN	0.0	NaN	NaN	N
1134	1974	1952	b'3 - BSc/MSc'	3560.0	2	0	-25.0	0.0	NaN	0.0	NaN	NaN	N
3166	1995	1981	b'2 - High School'	1813.0	4	1	-25.0	0.0	NaN	0.0	NaN	NaN	N
4023	1995	1988	b'3 - BSc/MSc'	1296.0	3	1	-25.0	0.0	NaN	0.0	NaN	NaN	N
4114	1991	1993	b'2 - High School'	1073.0	1	1	-25.0	0.0	NaN	0.0	NaN	NaN	N
4272	1983	1988	b'3 - BSc/MSc'	1458.0	1	1	-25.0	0.0	NaN	0.0	NaN	NaN	N
5984	1986	1959	b'3 - BSc/MSc'	2745.0	3	0	-25.0	0.0	NaN	0.0	NaN	NaN	N
6440	1991	1956	b'3 - BSc/MSc'	2375.0	2	1	-25.0	0.0	NaN	0.0	NaN	NaN	N
6615	1995	1997	b'1 - Basic'	1231.0	1	1	-25.0	0.0	NaN	0.0	NaN	NaN	N
8586	1997	1950	b'3 - BSc/MSc'	3269.0	1	0	-25.0	0.0	NaN	0.0	NaN	NaN	N
9399	1974	1956	b'1 - Basic'	2170.0	3	0	-25.0	0.0	NaN	0.0	NaN	NaN	N
9789	1976	1937	b'3 - BSc/MSc'	4050.0	1	0	-25.0	0.0	NaN	0.0	NaN	NaN	N

In [59]:
```
df_no_outliers.loc[(df_no_outliers['CustMonVal']==-25) & (df_no_outliers['ClaimsRate']==1)]
#CustMonVal = (annual profit from the customer) X (number of years that they are a customer) - (acquisition cost)
#acquisition cost = 25, because ClaimsRate = 1 (Amount paid by the insurance company (€)(x)/ Premiums (€)(x)) - note 2 years
```

Out[59]:

CustID	FirstPolYear	BirthYear	EducDeg	MonthSal	GeoLivArea	Children	CustMonVal	ClaimsRate	PremMotor	PremHousehold	PremHealth	PremLife	PremW
67	1984	1968	b'3 - BSc/MSc'	2544.0	3	1	-25.0	1.0	366.07	78.35	165.81	16.56	7
73	1990	1951	b'3 - BSc/MSc'	3206.0	4	1	-25.0	1.0	213.15	217.25	242.60	56.90	59
75	1985	1980	b'2 - High School'	1440.0	1	1	-25.0	1.0	436.75	65.00	140.47	12.89	(
96	1976	1986	b'3 - BSc/MSc'	2668.0	4	1	-25.0	1.0	296.50	70.00	200.15	60.79	8
103	1981	1980	b'2 - High School'	2071.0	4	1	-25.0	1.0	190.59	317.85	188.37	6.89	118
...	...	...	...	...	...	...	...	...	...	...	...	...	
9843	1975	1997	b'1 - Basic'	1155.0	3	0	-25.0	1.0	67.90	789.05	215.04	11.78	145
9908	1978	1995	b'2 - High School'	979.0	2	1	-25.0	1.0	59.90	672.35	181.70	70.57	153
10093	1993	1985	b'4 - PhD'	2063.0	3	1	-25.0	1.0	477.20	9.45	104.13	7.78	9
10138	1982	1966	b'2 - High School'	2158.0	4	1	-25.0	1.0	403.52	28.35	122.80	14.78	49
10225	1996	1989	b'2 - High School'	1355.0	4	1	-25.0	1.0	151.03	848.50	178.81	35.34	59

260 rows × 13 columns

In [60]:
```
# Filled nan values in insurance expenses with 0
df_no_outliers['PremMotor'].fillna(0, inplace = True)
df_no_outliers['PremHealth'].fillna(0, inplace = True)
df_no_outliers['PremLife'].fillna(0, inplace = True)
df_no_outliers['PremWork'].fillna(0, inplace = True)
df_no_outliers['PremHousehold'].fillna(0, inplace = True)
```

In [61]:
```
df_no_outliers.loc[(df_no_outliers['CustMonVal']==-25) & (df_no_outliers['ClaimsRate']==0)]
#CustMonVal = (annual profit from the customer) X (number of years that they are a customer) - (acquisition cost)
#acquisition cost = 25, because ClaimsRate = 0 (Amount paid by the insurance company (€)(0)/ Premiums (€)(0)) - not our client in
```

Out[61]:

CustID	FirstPolYear	BirthYear	EducDeg	MonthSal	GeoLivArea	Children	CustMonVal	ClaimsRate	PremMotor	PremHousehold	PremHealth	PremLife	PremW
863	1987	1981	b'3 - BSc/MSc'	2127.0	1	1	-25.0	0.0	0.0	0.0	0.0	0.0	
1134	1974	1952	b'3 - BSc/MSc'	3560.0	2	0	-25.0	0.0	0.0	0.0	0.0	0.0	
3166	1995	1981	b'2 - High School'	1813.0	4	1	-25.0	0.0	0.0	0.0	0.0	0.0	
4023	1995	1988	b'3 - BSc/MSc'	1296.0	3	1	-25.0	0.0	0.0	0.0	0.0	0.0	
4114	1991	1993	b'2 - High School'	1073.0	1	1	-25.0	0.0	0.0	0.0	0.0	0.0	
4272	1983	1988	b'3 - BSc/MSc'	1458.0	1	1	-25.0	0.0	0.0	0.0	0.0	0.0	
5984	1986	1959	b'3 - BSc/MSc'	2745.0	3	0	-25.0	0.0	0.0	0.0	0.0	0.0	
6440	1991	1956	b'3 - BSc/MSc'	2375.0	2	1	-25.0	0.0	0.0	0.0	0.0	0.0	
6615	1995	1997	b'1 - Basic'	1231.0	1	1	-25.0	0.0	0.0	0.0	0.0	0.0	
8586	1997	1950	b'3 - BSc/MSc'	3269.0	1	0	-25.0	0.0	0.0	0.0	0.0	0.0	
9399	1974	1956	b'1 - Basic'	2170.0	3	0	-25.0	0.0	0.0	0.0	0.0	0.0	
9789	1976	1937	b'3 - BSc/MSc'	4050.0	1	0	-25.0	0.0	0.0	0.0	0.0	0.0	

In [62]:
```
df_no_outliers.isna().sum()
```

Out[62]:
```
FirstPolYear     0
BirthYear        0
EducDeg          0
MonthSal         0
GeoLivArea       0
Children         0
CustMonVal       0
ClaimsRate       0
PremMotor        0
PremHousehold    0
PremHealth       0
PremLife         0
PremWork         0
dtype: int64
```

Visualization

In [63]:
```
import matplotlib.pyplot as plt
import seaborn as sns

def corrdot(*args, **kwargs):
    corr_r = args[0].corr(args[1], 'spearman')
    corr_text = f"{corr_r:2.2f}".replace("0.", ".")
    ax = plt.gca()
    ax.set_axis_off()
    marker_size = abs(corr_r) * 10000
    ax.scatter([.5], [.5], marker_size, [corr_r], alpha=0.6, cmap="coolwarm",
               vmin=-1, vmax=1, transform=ax.transAxes)
    font_size = abs(corr_r) * 40 + 5
    ax.annotate(corr_text, [.5, .5,],  xycoords="axes fraction",
                ha='center', va='center', fontsize=font_size)
```

In [64]:
```python
sns.set(style='white', font_scale=1.6)
g = sns.PairGrid(df_no_outliers, aspect=1.4, diag_sharey=False)
g.map_lower(sns.regplot, lowess=True, ci=False, line_kws={'color': 'black'})
g.map_diag(sns.distplot, kde_kws={'color': 'black'})
g.map_upper(corrdot)
```

Out[64]: <seaborn.axisgrid.PairGrid at 0x21502e8fe20>

Coherence

Back to Index

In [65]:
```python
df_coherence = df_no_outliers.copy()
```

In [66]:
```python
df_coherence.describe().T
```

Out[66]:

	count	mean	std	min	25%	50%	75%	max
FirstPolYear	10293.0	1986.017779	6.602504	1974.00	1980.00	1986.00	1992.00	1998.00
BirthYear	10293.0	1967.950938	17.241349	1935.00	1953.00	1968.00	1983.00	1999.00
MonthSal	10293.0	2498.369863	982.734760	333.00	1707.00	2501.00	3288.00	5021.00
GeoLivArea	10293.0	2.709608	1.266286	1.00	1.00	3.00	4.00	4.00
Children	10293.0	0.707374	0.454990	0.00	0.00	1.00	1.00	1.00
CustMonVal	10293.0	209.216637	240.038347	-490.20	-9.00	183.82	392.86	1010.74
ClaimsRate	10293.0	0.679853	0.318906	0.00	0.39	0.72	0.98	1.62
PremMotor	10293.0	295.877521	139.099152	-4.11	189.59	298.28	407.41	585.22
PremHousehold	10293.0	202.600661	225.492969	-75.00	48.90	132.25	285.60	1310.80
PremHealth	10293.0	166.401156	74.593831	-2.11	110.02	161.14	218.82	381.96
PremLife	10293.0	40.850792	45.895204	-7.00	9.78	24.67	56.90	284.61
PremWork	10293.0	40.136050	45.318805	-12.00	9.89	25.45	55.90	298.50

For a large dataset, vectorization of the code gave the best performance improvement of around 700x over the apply function. So, it is best practice to use vectorization for manipulating a large dataset. -https://towardsdatascience.com/dont-use-apply-in-python-there-are-better-alternatives-dc6364968f44 (https://towardsdatascience.com/dont-use-apply-in-python-there-are-better-alternatives-dc6364968f44)

Birth Year

In [67]:
```
df_coherence.loc[(df_coherence['BirthYear']<=1897) | (df_coherence['BirthYear']>=2000)].value_counts().sum()
```
Out[67]: 0

Birth Year and FirstPolYear

In [68]:
```
df_coherence.loc[df_coherence['BirthYear']+16>df_coherence['FirstPolYear']].value_counts().sum()
```
Out[68]: 4709

In [69]:
```
df_coherence.loc[df_coherence['BirthYear']+16>df_coherence['FirstPolYear']].head()
```
Out[69]:

CustID	FirstPolYear	BirthYear	EducDeg	MonthSal	GeoLivArea	Children	CustMonVal	ClaimsRate	PremMotor	PremHousehold	PremHealth	PremLife	PremW
1	1985	1982	b'2 - High School'	2177.0	1	1	380.97	0.39	375.85	79.45	146.36	47.01	16
2	1981	1995	b'2 - High School'	677.0	4	1	-131.13	1.12	77.46	416.20	116.69	194.48	106
4	1990	1981	b'3 - BSc/MSc'	1099.0	4	1	-16.99	0.99	182.48	43.35	311.17	35.34	28
5	1986	1973	b'3 - BSc/MSc'	1763.0	4	1	35.23	0.90	338.62	47.80	182.59	18.78	41
8	1988	1974	b'2 - High School'	1743.0	4	1	-144.91	1.13	248.27	397.30	144.36	66.68	53

In [70]:
```
pd.crosstab(df_coherence['BirthYear']==1998,df_coherence['EducDeg'])
```
Out[70]:

BirthYear \ EducDeg	b'1 - Basic'	b'2 - High School'	b'3 - BSc/MSc'	b'4 - PhD'
False	1204	3481	4816	698
True	68	26	0	0

In [71]:
```
pd.crosstab(df_coherence['BirthYear']>=1998,df_coherence['Children'])
```
Out[71]:

BirthYear \ Children	0	1
False	2978	7152
True	34	129

In [72]:
```
pd.crosstab(df_coherence['BirthYear']>=1996,df_coherence['MonthSal']>1500)
```
Out[72]:

BirthYear \ MonthSal	False	True
False	1532	8328
True	401	32

In [73]:
```
pd.crosstab(df_coherence['BirthYear']>=1998,df_coherence['PremHousehold']!=0)
```
Out[73]:

BirthYear \ PremHousehold	False	True
False	84	10046
True	6	157

In [74]:
```
#we dont have way to check coherence of FirstPolicyYear and we checked the coherence of birth year and seems okay. Well delete Fi
```

In [75]:
```
#FirstPolYear is WRONG
df_coherence.drop('FirstPolYear', axis=1, inplace=True)
```

In [76]:
```
df_coherence
```

Out[76]:

CustID	BirthYear	EducDeg	MonthSal	GeoLivArea	Children	CustMonVal	ClaimsRate	PremMotor	PremHousehold	PremHealth	PremLife	PremWork
1	1982	b'2 - High School'	2177.0	1	1	380.97	0.39	375.85	79.45	146.36	47.01	16.89
2	1995	b'2 - High School'	677.0	4	1	-131.13	1.12	77.46	416.20	116.69	194.48	106.13
3	1970	b'1 - Basic'	2277.0	3	0	504.67	0.28	206.15	224.50	124.58	86.35	99.02
4	1981	b'3 - BSc/MSc'	1099.0	4	1	-16.99	0.99	182.48	43.35	311.17	35.34	28.34
5	1973	b'3 - BSc/MSc'	1763.0	4	1	35.23	0.90	338.62	47.80	182.59	18.78	41.45
...	...	...	...	...	...	...	...	...	...	...	...	...
10292	1949	b'4 - PhD'	3188.0	2	0	-0.11	0.96	393.74	49.45	173.81	9.78	14.78
10293	1952	b'1 - Basic'	2431.0	3	0	183.82	0.00	133.58	1035.75	143.25	12.89	105.13
10294	1976	b'3 - BSc/MSc'	2918.0	1	1	524.10	0.21	403.63	132.80	142.25	12.67	4.89
10295	1977	b'1 - Basic'	1971.0	2	1	250.05	0.65	188.59	211.15	198.37	63.90	112.91
10296	1981	b'4 - PhD'	2815.0	1	1	463.75	0.27	414.08	94.45	141.25	6.89	12.89

10293 rows × 12 columns

In [77]:
```
#We assume that FirstPolYear is wrong, since the rest of the data become consistent if we ignore it. This is because if
#the other way around (assuming BirthYear is wrong) we would still have probably incoherent data like
# salary, children, education
```

Birth Year and Education

In [78]:
```
#b'1 - Basic' & <14
```

In [79]:
```
pd.crosstab(df_coherence['BirthYear']>2002,df_coherence['EducDeg'])
```

Out[79]:

BirthYear \ EducDeg	b'1 - Basic'	b'2 - High School'	b'3 - BSc/MSc'	b'4 - PhD'
False	1272	3507	4816	698

In [80]:
```
#b'2 - High School' & <18
```

In [81]:
```
pd.crosstab(df_coherence['BirthYear']>1998,df_coherence['EducDeg'])
```

Out[81]:

BirthYear \ EducDeg	b'1 - Basic'	b'2 - High School'	b'3 - BSc/MSc'	b'4 - PhD'
False	1203	3507	4816	698
True	69	0	0	0

In [82]:
```
#b'3 - BSc/MSc' & < 20
```

In [83]:
```
pd.crosstab(df_coherence['BirthYear']>1996,df_coherence['EducDeg'])
```

Out[83]:

BirthYear \ EducDeg	b'1 - Basic'	b'2 - High School'	b'3 - BSc/MSc'	b'4 - PhD'
False	1057	3426	4816	698
True	215	81	0	0

In [84]:
```
#b'4 - PhD' & < 23
```

In [85]:
```
pd.crosstab(df_coherence['BirthYear']>1993,df_coherence['EducDeg'])
```

Out[85]:

BirthYear \ EducDeg	b'1 - Basic'	b'2 - High School'	b'3 - BSc/MSc'	b'4 - PhD'
False	854	3242	4734	698
True	418	265	82	0

In [86]:
```
df_coherence.isnull().sum()
```

Out[86]:
```
BirthYear        0
EducDeg          0
MonthSal         0
GeoLivArea       0
Children         0
CustMonVal       0
ClaimsRate       0
PremMotor        0
PremHousehold    0
PremHealth       0
PremLife         0
PremWork         0
dtype: int64
```

Skewness

In [87]:
```
df_skew = df_coherence.copy()
```

In [88]:
```
non_metric_features = ["Children","EducDeg",'GeoLivArea']
metric_features = df_skew.columns.drop(non_metric_features).to_list()
metric_features
```

Out[88]:
```
['BirthYear',
 'MonthSal',
 'CustMonVal',
 'ClaimsRate',
 'PremMotor',
 'PremHousehold',
 'PremHealth',
 'PremLife',
 'PremWork']
```

In [89]:
```
from sklearn import preprocessing
```

In [90]:
```
scaler = preprocessing.MinMaxScaler()
```

In [91]:
```
scaled_feat = scaler.fit_transform(df_skew[metric_features])
df_skew[metric_features] = scaled_feat
```

In [92]:
```
df_skew.describe().T
```

Out[92]:

	count	mean	std	min	25%	50%	75%	max
BirthYear	10293.0	0.514858	0.269396	0.0	0.281250	0.515625	0.750000	1.0
MonthSal	10293.0	0.461896	0.209628	0.0	0.293089	0.462457	0.630333	1.0
GeoLivArea	10293.0	2.709608	1.266286	1.0	1.000000	3.000000	4.000000	4.0
Children	10293.0	0.707374	0.454990	0.0	0.000000	1.000000	1.000000	1.0
CustMonVal	10293.0	0.465986	0.159925	0.0	0.320599	0.449065	0.588338	1.0
ClaimsRate	10293.0	0.419663	0.196856	0.0	0.240741	0.444444	0.604938	1.0
PremMotor	10293.0	0.509031	0.236029	0.0	0.328678	0.513108	0.698284	1.0
PremHousehold	10293.0	0.200318	0.162717	0.0	0.089407	0.149553	0.260211	1.0
PremHealth	10293.0	0.438751	0.194219	0.0	0.291952	0.425053	0.575234	1.0
PremLife	10293.0	0.164092	0.157386	0.0	0.057543	0.108604	0.219128	1.0
PremWork	10293.0	0.167910	0.145954	0.0	0.070499	0.120612	0.218680	1.0

In [93]:
```
df_skew.isna().sum()
```

Out[93]:
```
BirthYear        0
EducDeg          0
MonthSal         0
GeoLivArea       0
Children         0
CustMonVal       0
ClaimsRate       0
PremMotor        0
PremHousehold    0
PremHealth       0
PremLife         0
PremWork         0
dtype: int64
```

In [94]:
```
df_skew[metric_features].skew()
#we need values between -0.5 and 0.5
```

Out[94]:
```
BirthYear        0.010330
MonthSal         0.003802
CustMonVal       0.504321
ClaimsRate      -0.236270
PremMotor       -0.092966
PremHousehold    1.654084
PremHealth       0.242987
PremLife         1.908196
PremWork         1.924430
dtype: float64
```

In [95]:
```
#df_skew['PremHousehold'][df_skew['PremHousehold'] > 0].min()
```

In [96]:
```
# df_skew['sqrt_CustMonVal']=np.sqrt(df_skew['CustMonVal'])
```

In [97]:
```
# df_skew['log_CustMonVal']=np.sqrt(df_skew['CustMonVal'])
```

In [98]:
```
df_skew['BoxCox_CustMonVal'],parameters=stats.boxcox(df_skew['CustMonVal']+df_skew['CustMonVal'][df_skew['CustMonVal'] > 0].min()
```

In [99]:
```
df_skew.drop('CustMonVal', axis=1, inplace=True)
```

In [100]:
```
# df_skew['sqrt_household']=np.sqrt(df_skew['PremHousehold'])
```

In [101]:
```
# df_skew['log_household']=np.log10(df_skew['PremHousehold']+df_skew['PremHousehold'][df_skew['PremHousehold'] > 0].min())
```

In [102]:
```
df_skew['BoxCox_PremHousehold'],parameters=stats.boxcox(df_skew['PremHousehold']+df_skew['PremHousehold'][df_skew['PremHousehold'
```

In [103]:
```
df_skew.drop('PremHousehold', axis=1, inplace=True)
```

In [104]:
```
# df_skew['sqrt_life']=np.sqrt(df_skew['PremLife'])
```

In [105]:
```
# df_skew['log_life']=np.log10(df_skew['PremLife']+df_skew['PremLife'][df_skew['PremLife'] > 0].min())
```

In [106]:
```
df_skew['BoxCox_PremLife'],parameters=stats.boxcox(df_skew['PremLife']+df_skew['PremLife'][df_skew['PremLife'] > 0].min())
```

In [107]:
```
df_skew.drop('PremLife', axis=1, inplace=True)
```

In [108]:
```
# df_skew['sqrt_work']=np.sqrt(df_skew['PremWork'])
```

In [109]:
```
# df_skew['log_work']=np.log10(df_skew['PremWork']+df_skew['PremWork'][df_skew['PremWork'] > 0].min())
```

In [110]:
```
df_skew['BoxCox_PremWork'],parameters=stats.boxcox(df_skew['PremWork']+df_skew['PremWork'][df_skew['PremWork'] > 0].min())
```

In [111]:
```
df_skew.drop('PremWork', axis=1, inplace=True)
```

In [112]:
```
df_skew['BoxCox_PremMotor'],parameters=stats.boxcox(df_skew['PremMotor']+df_skew['PremMotor'][df_skew['PremMotor'] > 0].min())
```

In [113]:
```
df_skew.drop('PremMotor', axis=1, inplace=True)
```

In [114]:
```
df_skew['BoxCox_PremHealth'],parameters=stats.boxcox(df_skew['PremHealth']+df_skew['PremHealth'][df_skew['PremHealth'] > 0].min()
```

In [115]:
```
df_skew.drop('PremHealth', axis=1, inplace=True)
```

In [116]:
```
# df_skew['BoxCox_%SalaryInsurance'],parameters=stats.boxcox(df_skew['%SalaryInsurance']+df_skew['%SalaryInsurance'][df_skew['%Sa
```

In [117]:
```
# df_skew['log_%SalaryInsurance']=np.log10(df_skew['%SalaryInsurance']+df_skew['%SalaryInsurance'][df_skew['%SalaryInsurance'] >
```

In [118]:
```
# df_skew['sqrt_%SalaryInsurance']=np.sqrt(df_skew['%SalaryInsurance'])
```

In [119]:
```
# df_skew['BoxCox_%SalaryInsuranceHealth'],parameters=stats.boxcox(df_skew['%SalaryInsuranceHealth']+df_skew['%SalaryInsuranceHea
```

In [120]:
```
# df_skew['BoxCox_%SalaryInsuranceHousehold'],parameters=stats.boxcox(df_skew['%SalaryInsuranceHousehold']+df_skew['%SalaryInsura
```

In [121]:
```
# df_skew['BoxCox_%SalaryInsuranceLife'],parameters=stats.boxcox(df_skew['%SalaryInsuranceLife']+df_skew['%SalaryInsuranceLife'][
```

In [122]:
```
# df_skew['BoxCox_%SalaryInsuranceMotor'],parameters=stats.boxcox(df_skew['%SalaryInsuranceMotor']+df_skew['%SalaryInsuranceMotor
```

In [123]:
```
# df_skew['BoxCox_%SalaryInsuranceWork'],parameters=stats.boxcox(df_skew['%SalaryInsuranceWork']+df_skew['%SalaryInsuranceWork'][
```

In [124]:
```
# df_skew['BoxCox_TotalPremiums'],parameters=stats.boxcox(df_skew['TotalPremiums']+df_skew['TotalPremiums'][df_skew['TotalPremium
```

In [125]:
```
non_metric_features = ["Children","EducDeg",'GeoLivArea']
metric_features = df_skew.columns.drop(non_metric_features).to_list()
metric_features
```

Out[125]:
```
['BirthYear',
 'MonthSal',
 'ClaimsRate',
 'BoxCox_CustMonVal',
 'BoxCox_PremHousehold',
 'BoxCox_PremLife',
 'BoxCox_PremWork',
 'BoxCox_PremMotor',
 'BoxCox_PremHealth']
```

In [126]:
```
df_skew[metric_features].skew()
#we need values between -0.5 and 0.5
```

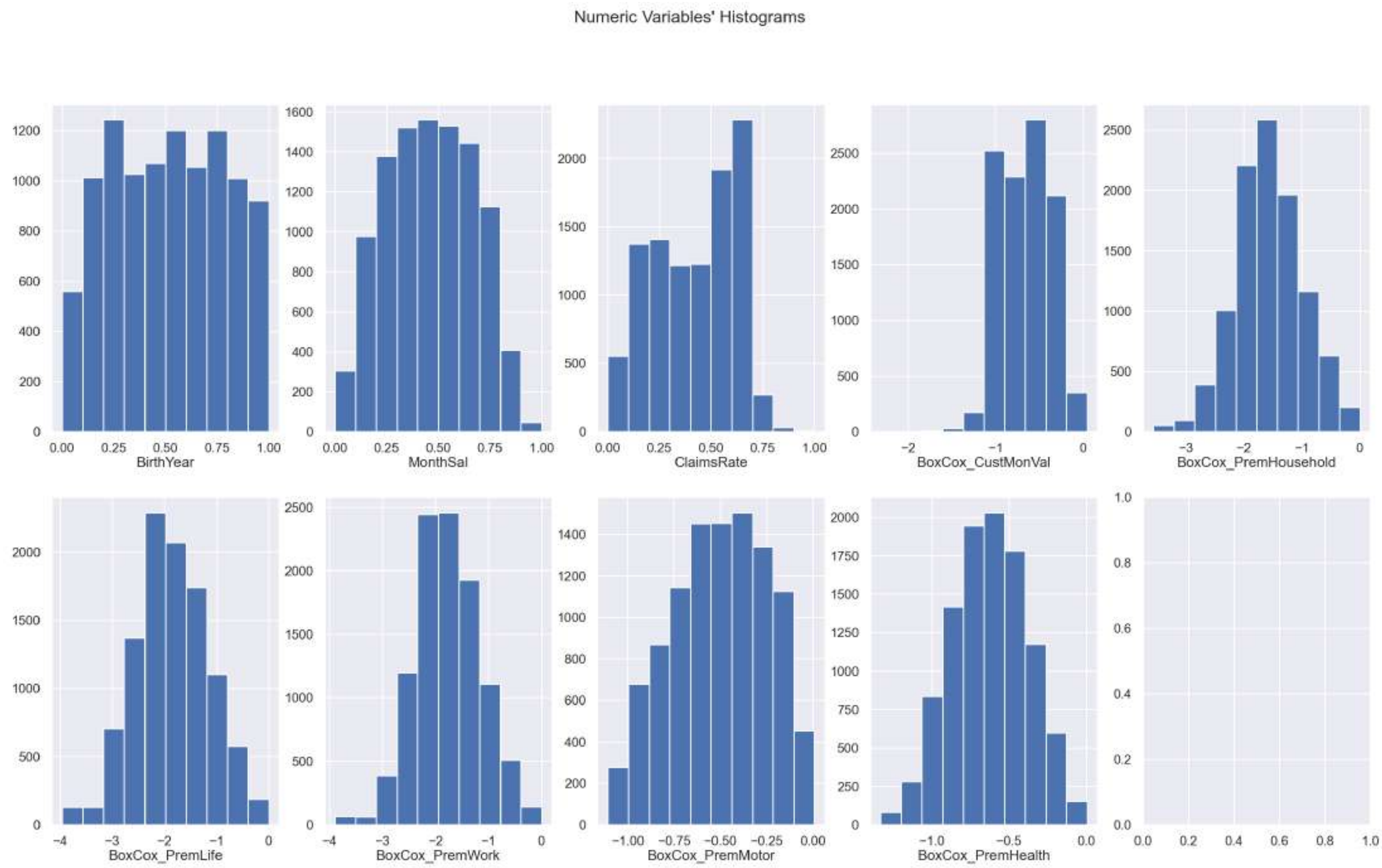
Out[126]:
```
BirthYear               0.010330
MonthSal                0.003802
ClaimsRate             -0.236270
BoxCox_CustMonVal      -0.008580
BoxCox_PremHousehold    0.006925
BoxCox_PremLife        -0.002318
BoxCox_PremWork         0.010292
BoxCox_PremMotor       -0.194695
BoxCox_PremHealth      -0.061862
dtype: float64
```

In [127]:
```
df_skew[metric_features].kurtosis()
```

Out[127]:
```
BirthYear              -1.154698
MonthSal               -0.910507
ClaimsRate             -1.161314
BoxCox_CustMonVal      -0.497057
BoxCox_PremHousehold    0.071065
BoxCox_PremLife        -0.105390
BoxCox_PremWork         0.176562
BoxCox_PremMotor       -0.837329
BoxCox_PremHealth      -0.407465
dtype: float64
```

In [128]:

```python
# All Numeric Variables' Histograms in one figure
sns.set()

# Prepare figure. Create individual axes where each histogram will be placed
fig, axes = plt.subplots(2, ceil(len(metric_features) / 2), figsize=(20, 11))

# Plot data
# Iterate across axes objects and associate each histogram (hint: use the ax.hist() instead of plt.hist()):
for ax, feat in zip(axes.flatten(), metric_features): # Notice the zip() function and flatten() method
    ax.hist(df_skew[feat][~np.isnan(df_skew[feat])])
    ax.set_title(feat, y=-0.13)

# Layout
# Add a centered title to the figure:
title = "Numeric Variables' Histograms"

plt.suptitle(title)

plt.show()
```

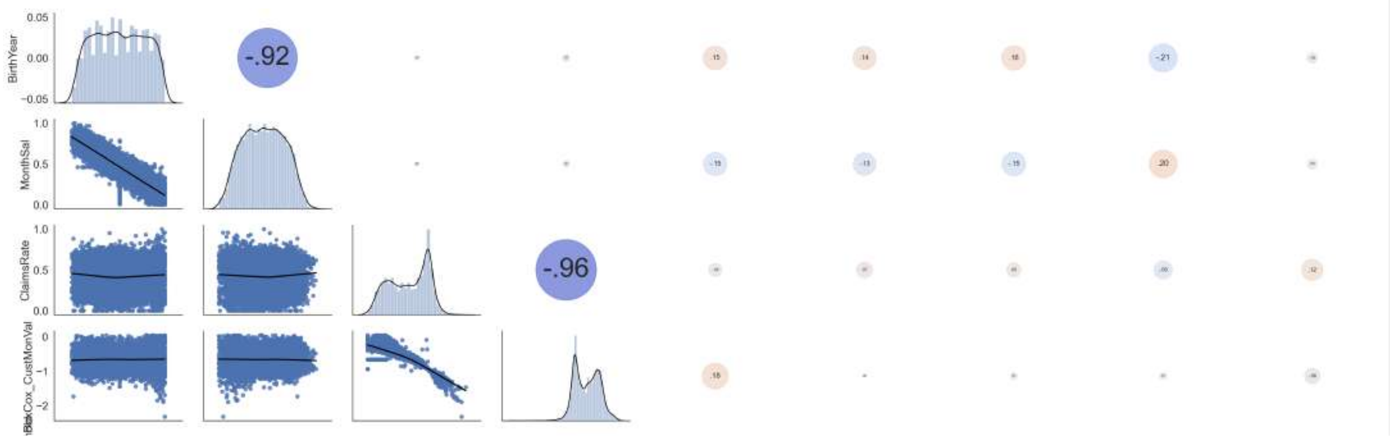


In [129]:

```python
sns.set(style='white', font_scale=1.6)
data = df_skew[metric_features]
graph = sns.PairGrid(data, aspect=1.4, diag_sharey=False)
graph.map_lower(sns.regplot, lowess=True, ci=False, line_kws={'color': 'black'})
graph.map_diag(sns.distplot, kde_kws={'color': 'black'})
graph.map_upper(corrdot)
```

Out[129]:

```
<seaborn.axisgrid.PairGrid at 0x21579dafb20>
```



New Variables

In [130]:
```python
df_variables = df_skew.copy()
```

Age

In [131]:
```python
df_variables['Age'] = 2016 - df_coherence['BirthYear']
```

In [132]:
```python
df_variables.drop('BirthYear', axis=1, inplace=True)
```

Total Premiums (Annual)

In [133]:
```python
df_variables['BoxCox_TotalPremiums'] = (df_variables['BoxCox_PremHealth'] + df_variables['BoxCox_PremHousehold'] + df_variables['
+ df_variables['BoxCox_PremMotor'] + df_variables['BoxCox_PremWork'])
```

Amount paid by the insurance company (annual)

In [134]:
```python
# df_final['ClaimsRate'] = Amount paid by the insurance company (€)/ Premiums (€) Note: in the last 2 years
```

In [135]:
```python
#df_final['AmountPaid'] = ((df_final['TotalPremiums']*2)*df_final['ClaimsRate'])/2
```

Annual profit from the customer

In [136]:
```python
#df_final['Profit'] = np.where((df_final['TotalPremiums'] - df_final['AmountPaid']) <= 0 , 0, (df_final['TotalPremiums'] - df_fin
```

Years as a Customer (number of years that they are a customer)

In [137]:
```python
# Lifetime value = (annual profit from the customer) X (number of years that they are a customer) - (acquisition cost)
```

In [138]:
```python
#acquisition cost = 25
#annual profit from the customer = df_final['Profit']
# Lifetime Value = df_final['CustMonValue']
```

In [139]:
```python
#df_final['YearsCustomer'] = np.where(((df_final['Profit'] <= 0)),0,((df_final['CustMonVal']+25)/df_final['Profit']))
```

FirstPolYear

In [140]:
```python
#df_final['FirstPolYearCorrect'] = (2016-df_final['Age']) + df_final['YearsCustomer']
```

Annual Salary (12x)

In [141]:
```python
df_variables['Salary'] = df_coherence['MonthSal']*12
```

In [142]:
```python
df_variables['Salary'][df_variables['Salary'] > 0].min()
```

Out[142]:
```
3996.0
```

In [143]:
```python
df_variables['BoxCox_Salary'],parameters=stats.boxcox(df_variables['Salary'])
```

%SalaryInsurance

In [144]:
```python
df_variables['BoxCox_Salary%SalaryInsurance'] = df_variables['BoxCox_TotalPremiums']/df_variables['BoxCox_Salary']
```

In [145]:
```python
df_variables['BoxCox_Salary%SalaryInsurance']
```

Out[145]:
```
CustID
1        -0.001079
2        -0.001766
3        -0.000781
4        -0.001906
5        -0.001328
            ...
10292    -0.000898
10293    -0.000753
10294    -0.000959
10295    -0.000873
10296    -0.001001
Name: BoxCox_Salary%SalaryInsurance, Length: 10293, dtype: float64
```

%SalaryInsuranceMotor

In [146]:
```
df_variables['BoxCox_Salary%SalaryInsuranceMotor'] =  np.where(df_variables['BoxCox_TotalPremiums']==0,0,((df_variables['BoxCox_P
```

In [147]:
```
df_variables['BoxCox_Salary%SalaryInsuranceMotor']
```

Out[147]:
```
CustID
1       -0.000061
2       -0.000418
3       -0.000110
4       -0.000215
5       -0.000086
           ...
10292   -0.000040
10293   -0.000126
10294   -0.000041
10295   -0.000130
10296   -0.000040
Name: BoxCox_Salary%SalaryInsuranceMotor, Length: 10293, dtype: float64
```

%SalaryInsuranceHousehold

In [148]:
```
df_variables['BoxCox_Salary%SalaryInsuranceHousehold'] =  np.where(df_variables['BoxCox_TotalPremiums']==0,0,((df_variables['BoxC
```

In [149]:
```
df_variables['BoxCox_Salary%SalaryInsuranceHousehold']
```

Out[149]:
```
CustID
1       -0.000308
2       -0.000426
3       -0.000219
4       -0.000596
5       -0.000397
           ...
10292   -0.000241
10293   -0.000033
10294   -0.000214
10295   -0.000254
10296   -0.000240
Name: BoxCox_Salary%SalaryInsuranceHousehold, Length: 10293, dtype: float64
```

%SalaryInsuranceHealth

In [150]:
```
df_variables['BoxCox_Salary%SalaryInsuranceHealth'] =  np.where(df_variables['BoxCox_TotalPremiums']==0,0,((df_variables['BoxCox_
```

In [151]:
```
df_variables['BoxCox_Salary%SalaryInsuranceHealth']
```

Out[151]:
```
CustID
1       -0.000116
2       -0.000354
3       -0.000124
4       -0.000056
5       -0.000114
           ...
10292   -0.000073
10293   -0.000107
10294   -0.000093
10295   -0.000095
10296   -0.000096
Name: BoxCox_Salary%SalaryInsuranceHealth, Length: 10293, dtype: float64
```

%SalaryInsuranceLife

In [152]:
```
df_variables['BoxCox_Salary%SalaryInsuranceLife'] =  np.where(df_variables['BoxCox_TotalPremiums']==0,0,((df_variables['BoxCox_Pr
```

In [153]:
```
df_variables['BoxCox_Salary%SalaryInsuranceLife']
```
Out[153]:
```
CustID
1        -0.000255
2        -0.000162
3        -0.000172
4        -0.000509
5        -0.000416
            ...
10292    -0.000290
10293    -0.000346
10294    -0.000298
10295    -0.000237
10296    -0.000338
Name: BoxCox_Salary%SalaryInsuranceLife, Length: 10293, dtype: float64
```

%SalaryInsuranceWork

In [154]:
```
df_variables['BoxCox_Salary%SalaryInsuranceWork'] =  np.where(df_variables['BoxCox_TotalPremiums']==0,0,((df_variables['BoxCox_Pr
```
In [155]:
```
df_variables['BoxCox_Salary%SalaryInsuranceWork']
```
Out[155]:
```
CustID
1        -0.000340
2        -0.000406
3        -0.000156
4        -0.000530
5        -0.000314
            ...
10292    -0.000253
10293    -0.000141
10294    -0.000313
10295    -0.000157
10296    -0.000288
Name: BoxCox_Salary%SalaryInsuranceWork, Length: 10293, dtype: float64
```

BinnedCR

In [156]:
```
#ClaimsRate = Amount paid by the insurance company (€)/ Premiums (€) [Last2Years]
```
In [157]:
```
df_variables['BinnedCR'] = np.where(df_coherence['ClaimsRate'] <= 0.25,4 ,np.where(df_coherence['ClaimsRate'] <= 0.80, 3
                                                                           ,np.where(df_coherence['ClaimsRate'] <= 1, 2,1)))
```

Liquid Salary

In [158]:
```
df_variables['BoxCox_LiquidSalary'] = df_variables['BoxCox_Salary'] - df_variables['BoxCox_TotalPremiums']
```

Minimum Wage

In [159]:
```
# df_variables['MinimumWage'] = np.where(
#     df_coherence['MonthSal'] > 618.33, 0, 1)
```
In [160]:
```
# df_variables['MinimumWage'].value_counts()
```

In [161]:
```
df_variables = df_variables[['Age','EducDeg','MonthSal','BoxCox_Salary','Children','GeoLivArea','BoxCox_PremMotor','BoxCox_Salary
df_variables
```

Out[161]:

CustID	Age	EducDeg	MonthSal	BoxCox_Salary	Children	GeoLivArea	BoxCox_PremMotor	BoxCox_Salary%SalaryInsuranceMotor	BoxCox_PremHousehold	Bo
1	34	b'2 - High School'	0.393345	5843.071899	1	1	-0.355902	-0.000061	-1.797062	
2	21	b'2 - High School'	0.073379	2202.416007	1	4	-0.920496	-0.000418	-0.939224	
3	46	b'1 - Basic'	0.414676	6066.410473	0	3	-0.666285	-0.000110	-1.329386	
4	35	b'3 - BSc/MSc'	0.163396	3301.259295	1	4	-0.711374	-0.000215	-1.968312	
5	43	b'3 - BSc/MSc'	0.305034	4899.238100	1	4	-0.422432	-0.000086	-1.945129	
...	...	...	...	...	...	...	...	...	...	
10292	67	b'4 - PhD'	0.609002	8035.252502	0	2	-0.324185	-0.000040	-1.936701	
10293	64	b'1 - Basic'	0.447526	6407.225920	0	3	-0.806571	-0.000126	-0.212651	
10294	40	b'3 - BSc/MSc'	0.551408	7462.788414	1	1	-0.306716	-0.000041	-1.595008	
10295	39	b'1 - Basic'	0.349403	5377.531714	1	2	-0.699680	-0.000130	-1.363548	
10296	35	b'4 - PhD'	0.529437	7242.133039	1	1	-0.288308	-0.000040	-1.735222	

10293 rows × 22 columns

In [162]:
```
df_variables.isna().sum()
```

Out[162]:
```
Age                                      0
EducDeg                                  0
MonthSal                                 0
BoxCox_Salary                            0
Children                                 0
GeoLivArea                               0
BoxCox_PremMotor                         0
BoxCox_Salary%SalaryInsuranceMotor       0
BoxCox_PremHousehold                     0
BoxCox_Salary%SalaryInsuranceHousehold   0
BoxCox_PremHealth                        0
BoxCox_Salary%SalaryInsuranceHealth      0
BoxCox_PremLife                          0
BoxCox_Salary%SalaryInsuranceLife        0
BoxCox_PremWork                          0
BoxCox_Salary%SalaryInsuranceWork        0
BoxCox_TotalPremiums                     0
BoxCox_Salary%SalaryInsurance            0
BoxCox_LiquidSalary                      0
BoxCox_CustMonVal                        0
ClaimsRate                               0
BinnedCR                                 0
dtype: int64
```

In [163]:
```
df_variables.describe().T
```

Out[163]:

	count	mean	std	min	25%	50%	75%	max
Age	10293.0	48.049062	17.241349	17.000000	33.000000	48.000000	63.000000	8.100000e+01
MonthSal	10293.0	0.461896	0.209628	0.000000	0.293089	0.462457	0.630333	1.000000e+00
BoxCox_Salary	10293.0	6479.350760	2181.739377	1217.286709	4768.911654	6560.954526	8245.217096	1.174226e+04
Children	10293.0	0.707374	0.454990	0.000000	0.000000	1.000000	1.000000	1.000000e+00
GeoLivArea	10293.0	2.709608	1.266286	1.000000	1.000000	3.000000	4.000000	4.000000e+00
BoxCox_PremMotor	10293.0	-0.506985	0.257391	-1.107975	-0.697770	-0.495402	-0.300052	6.971396e-03
BoxCox_Salary%SalaryInsuranceMotor	10293.0	-0.000100	0.000098	-0.000862	-0.000115	-0.000070	-0.000043	9.293687e-07
BoxCox_PremHousehold	10293.0	-1.566221	0.577859	-3.564087	-1.939501	-1.596865	-1.187202	3.602678e-03
BoxCox_Salary%SalaryInsuranceHousehold	10293.0	-0.000268	0.000139	-0.001783	-0.000335	-0.000246	-0.000174	9.761196e-07
BoxCox_PremHealth	10293.0	-0.626450	0.245901	-1.330623	-0.802442	-0.628162	-0.448029	5.489803e-03
BoxCox_Salary%SalaryInsuranceHealth	10293.0	-0.000112	0.000071	-0.000808	-0.000141	-0.000100	-0.000066	7.793765e-07
BoxCox_PremLife	10293.0	-1.852811	0.702563	-3.958565	-2.327941	-1.893965	-1.359374	3.424166e-03
BoxCox_Salary%SalaryInsuranceLife	10293.0	-0.000317	0.000167	-0.001742	-0.000398	-0.000291	-0.000202	1.239420e-06
BoxCox_PremWork	10293.0	-1.766428	0.617953	-3.895169	-2.171154	-1.802103	-1.352504	3.216187e-03
BoxCox_Salary%SalaryInsuranceWork	10293.0	-0.000303	0.000154	-0.001888	-0.000375	-0.000278	-0.000200	6.913628e-07
BoxCox_TotalPremiums	10293.0	-6.318895	1.377590	-11.507648	-7.303040	-6.262982	-5.287730	-3.354382e+00
BoxCox_Salary%SalaryInsurance	10293.0	-0.001101	0.000466	-0.004855	-0.001329	-0.001031	-0.000757	-3.499879e-04
BoxCox_LiquidSalary	10293.0	6485.669655	2182.009556	1221.074167	4777.952624	6567.681688	8251.728082	1.174757e+04
BoxCox_CustMonVal	10293.0	-0.659831	0.275538	-2.317761	-0.909978	-0.654447	-0.432443	4.799882e-02
ClaimsRate	10293.0	0.419663	0.196856	0.000000	0.240741	0.444444	0.604938	1.000000e+00
BinnedCR	10293.0	2.516468	0.920069	1.000000	2.000000	3.000000	3.000000	4.000000e+00

In [164]:
```
non_metric_features = ["Children","EducDeg",'GeoLivArea','BinnedCR']
metric_features = df_variables.columns.drop(non_metric_features).to_list()
metric_features
```

Out[164]:
```
['Age',
 'MonthSal',
 'BoxCox_Salary',
 'BoxCox_PremMotor',
 'BoxCox_Salary%SalaryInsuranceMotor',
 'BoxCox_PremHousehold',
 'BoxCox_Salary%SalaryInsuranceHousehold',
 'BoxCox_PremHealth',
 'BoxCox_Salary%SalaryInsuranceHealth',
 'BoxCox_PremLife',
 'BoxCox_Salary%SalaryInsuranceLife',
 'BoxCox_PremWork',
 'BoxCox_Salary%SalaryInsuranceWork',
 'BoxCox_TotalPremiums',
 'BoxCox_Salary%SalaryInsurance',
 'BoxCox_LiquidSalary',
 'BoxCox_CustMonVal',
 'ClaimsRate']
```

In [165]:
```
scaled_feat = scaler.fit_transform(df_variables[metric_features])
df_variables[metric_features] = scaled_feat
```

In [166]:
```
df_variables.describe().T
```

Out[166]:

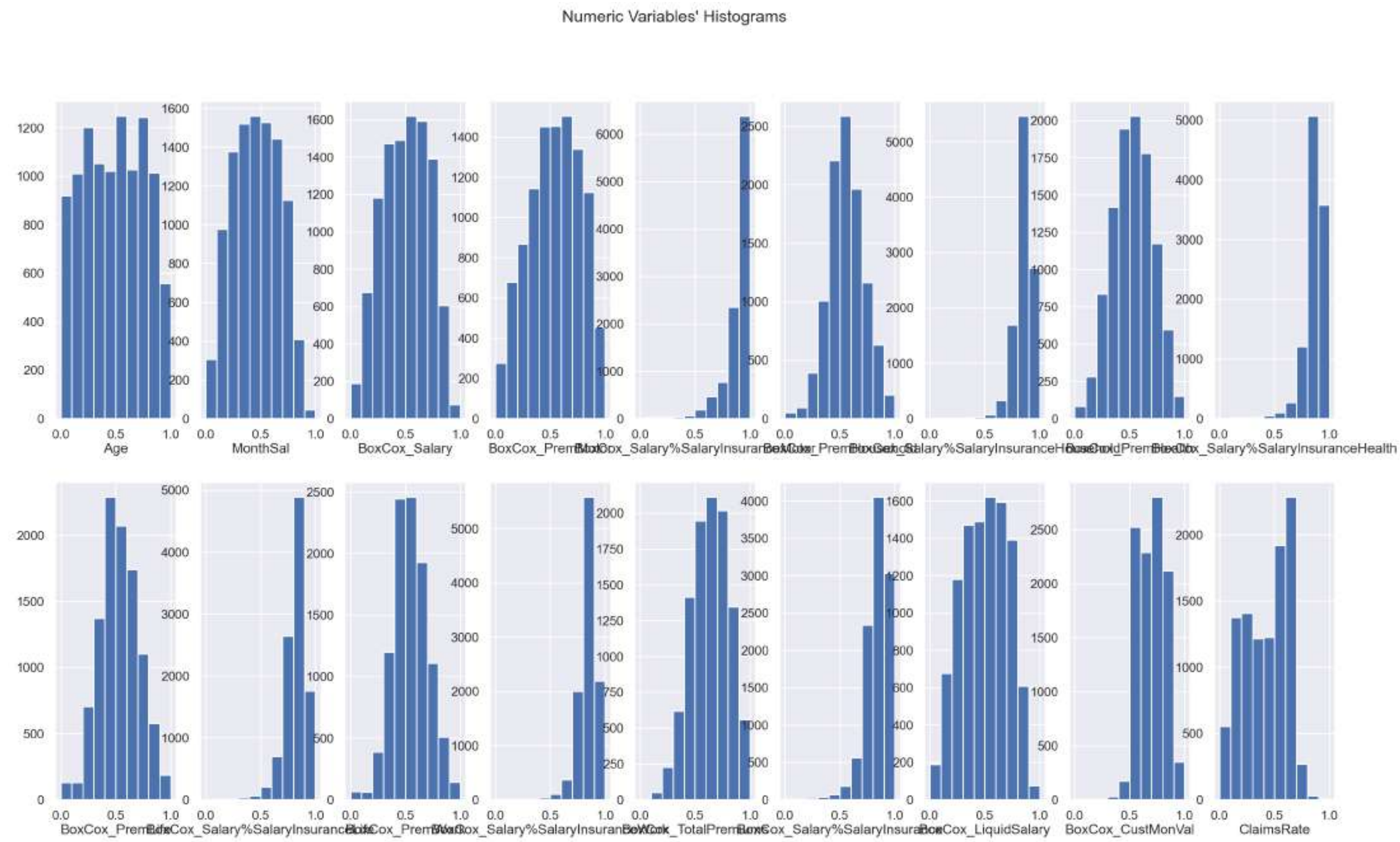
	count	mean	std	min	25%	50%	75%	max
Age	10293.0	0.485142	0.269396	0.0	0.250000	0.484375	0.718750	1.0
MonthSal	10293.0	0.461896	0.209628	0.0	0.293089	0.462457	0.630333	1.0
BoxCox_Salary	10293.0	0.499960	0.207292	0.0	0.337448	0.507713	0.667739	1.0
Children	10293.0	0.707374	0.454990	0.0	0.000000	1.000000	1.000000	1.0
GeoLivArea	10293.0	2.709608	1.266286	1.0	1.000000	3.000000	4.000000	4.0
BoxCox_PremMotor	10293.0	0.539031	0.230855	0.0	0.367915	0.549420	0.724629	1.0
BoxCox_Salary%SalaryInsuranceMotor	10293.0	0.882724	0.113358	0.0	0.865458	0.918122	0.949330	1.0
BoxCox_PremHousehold	10293.0	0.559989	0.161970	0.0	0.455361	0.551399	0.666225	1.0
BoxCox_Salary%SalaryInsuranceHousehold	10293.0	0.849068	0.077793	0.0	0.811526	0.861420	0.902021	1.0
BoxCox_PremHealth	10293.0	0.527031	0.184042	0.0	0.395312	0.525750	0.660568	1.0
BoxCox_Salary%SalaryInsuranceHealth	10293.0	0.860421	0.088288	0.0	0.825168	0.875197	0.917089	1.0
BoxCox_PremLife	10293.0	0.531489	0.177326	0.0	0.411567	0.521102	0.656032	1.0
BoxCox_Salary%SalaryInsuranceLife	10293.0	0.817182	0.095918	0.0	0.770836	0.832625	0.883235	1.0
BoxCox_PremWork	10293.0	0.546057	0.158515	0.0	0.442238	0.536906	0.652235	1.0
BoxCox_Salary%SalaryInsuranceWork	10293.0	0.839000	0.081398	0.0	0.801216	0.852590	0.893544	1.0
BoxCox_TotalPremiums	10293.0	0.636402	0.168962	0.0	0.515696	0.643260	0.762874	1.0
BoxCox_Salary%SalaryInsurance	10293.0	0.833176	0.103398	0.0	0.782760	0.848779	0.909568	1.0
BoxCox_LiquidSalary	10293.0	0.500128	0.207287	0.0	0.337898	0.507919	0.667900	1.0
BoxCox_CustMonVal	10293.0	0.700802	0.116469	0.0	0.595066	0.703078	0.796918	1.0
ClaimsRate	10293.0	0.419663	0.196856	0.0	0.240741	0.444444	0.604938	1.0
BinnedCR	10293.0	2.516468	0.920069	1.0	2.000000	3.000000	3.000000	4.0

In [167]:
```
non_metric_features = ["Children","EducDeg",'GeoLivArea','BinnedCR']
metric_features = df_variables.columns.drop(non_metric_features).to_list()
metric_features
```

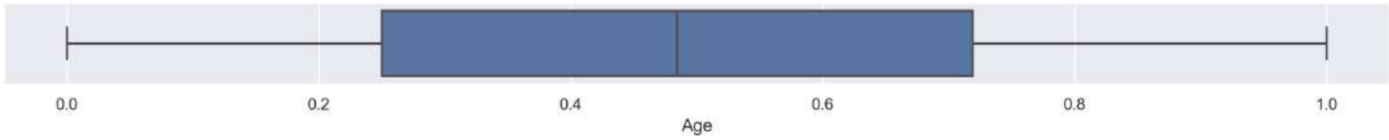
Out[167]:
```
['Age',
 'MonthSal',
 'BoxCox_Salary',
 'BoxCox_PremMotor',
 'BoxCox_Salary%SalaryInsuranceMotor',
 'BoxCox_PremHousehold',
 'BoxCox_Salary%SalaryInsuranceHousehold',
 'BoxCox_PremHealth',
 'BoxCox_Salary%SalaryInsuranceHealth',
 'BoxCox_PremLife',
 'BoxCox_Salary%SalaryInsuranceLife',
 'BoxCox_PremWork',
 'BoxCox_Salary%SalaryInsuranceWork',
 'BoxCox_TotalPremiums',
 'BoxCox_Salary%SalaryInsurance',
 'BoxCox_LiquidSalary',
 'BoxCox_CustMonVal',
 'ClaimsRate']
```

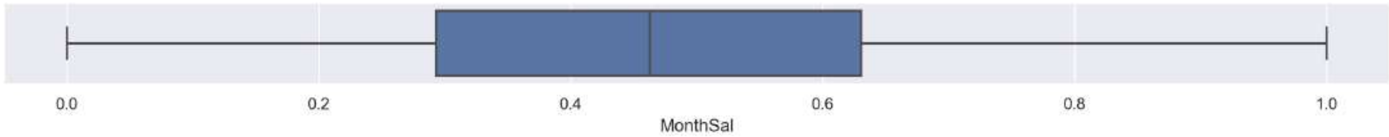
In [168]:

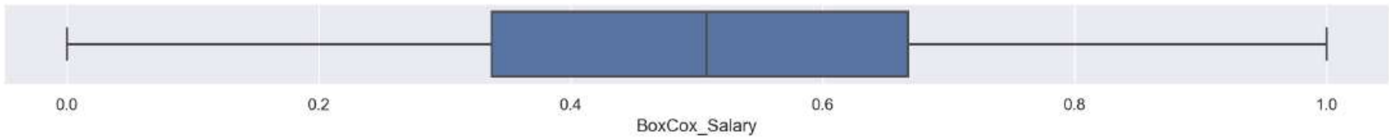
```
# All Numeric Variables' Histograms in one figure
sns.set()

# Prepare figure. Create individual axes where each histogram will be placed
fig, axes = plt.subplots(2, ceil(len(metric_features) / 2), figsize=(20, 11))

# Plot data
# Iterate across axes objects and associate each histogram (hint: use the ax.hist() instead of plt.hist()):
for ax, feat in zip(axes.flatten(), metric_features): # Notice the zip() function and flatten() method
    ax.hist(df_variables[feat][~np.isnan(df_variables[feat])])
    ax.set_title(feat, y=-0.13)

# Layout
# Add a centered title to the figure:
title = "Numeric Variables' Histograms"

plt.suptitle(title)

plt.show()
```

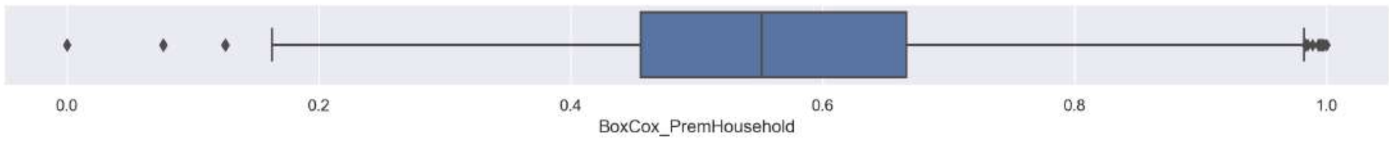


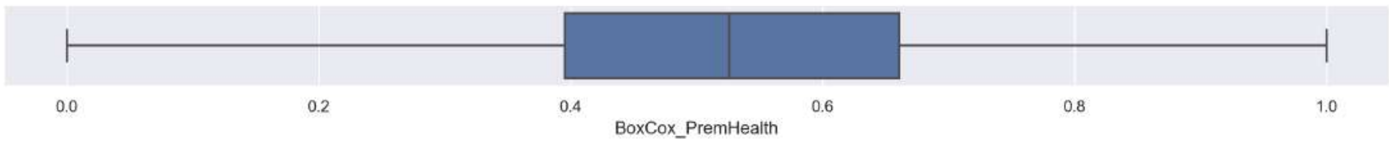
In [169]:
```python
for column in df_variables[metric_features]:
        plt.figure(figsize=(17,1))
        sns.boxplot(data=df_variables, x=column)
```

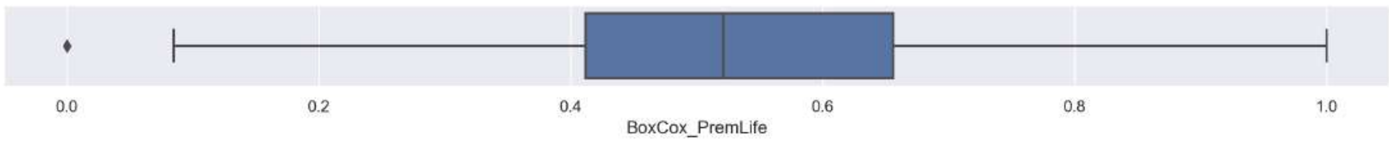


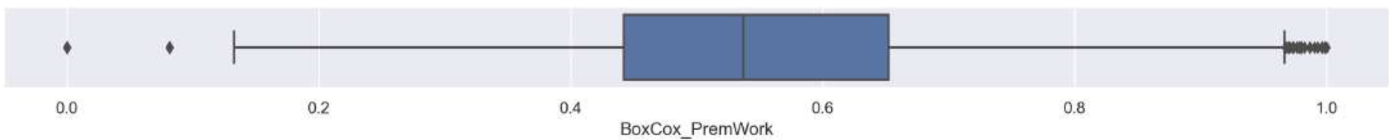


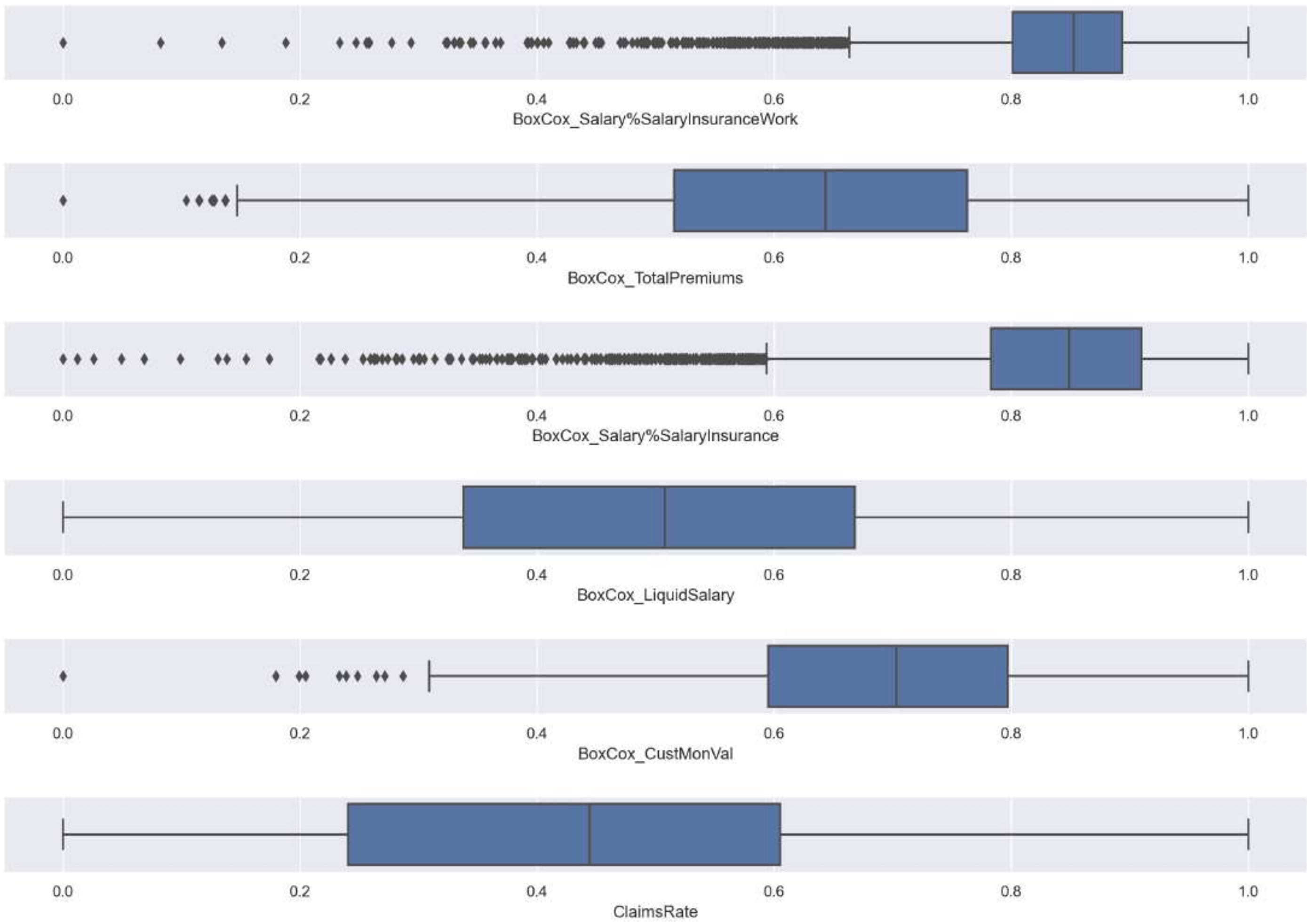












Encoders

Ordinal Encoder

In [170]:
```
df_final = df_variables.copy()
```

In [171]:
```
df_final['EducDeg'].value_counts()
```

Out[171]:
```
b'3 - BSc/MSc'        4816
b'2 - High School'    3507
b'1 - Basic'          1272
b'4 - PhD'             698
Name: EducDeg, dtype: int64
```

In [172]:
```
from sklearn.preprocessing import OrdinalEncoder

df_final_enc = df_final.copy()

EducationOrder = [[b'1 - Basic',b'2 - High School',b'3 - BSc/MSc',b'4 - PhD']]
encoder = OrdinalEncoder(categories=EducationOrder)
df_final_enc['EducDeg'] = encoder.fit_transform(df_final_enc[['EducDeg']])
df_final_enc['EducDeg'].value_counts()
```

Out[172]:
```
2.0    4816
1.0    3507
0.0    1272
3.0     698
Name: EducDeg, dtype: int64
```

In [173]:
```
df_final_enc
```

Out[173]:

	Age	EducDeg	MonthSal	BoxCox_Salary	Children	GeoLivArea	BoxCox_PremMotor	BoxCox_Salary%SalaryInsuranceMotor	BoxCox_PremHousehold
CustID									
1	0.265625	1.0	0.393345	0.439506	1	1	0.674537	0.928362	0.495286
2	0.062500	1.0	0.073379	0.093599	1	4	0.168151	0.514749	0.735732
3	0.453125	0.0	0.414676	0.460726	0	3	0.396153	0.871688	0.626372
4	0.281250	2.0	0.163396	0.198003	1	4	0.355713	0.749293	0.447285
5	0.406250	2.0	0.305034	0.349830	1	4	0.614866	0.899037	0.453783
...	...	...	...	...	...	...	...	...	..
10292	0.781250	3.0	0.609002	0.647790	0	2	0.702985	0.952185	0.456146
10293	0.734375	0.0	0.447526	0.493107	0	3	0.270331	0.853092	0.939385
10294	0.359375	2.0	0.551408	0.593399	1	1	0.718652	0.951311	0.551920
10295	0.343750	0.0	0.349403	0.395274	1	2	0.366201	0.848195	0.616797
10296	0.281250	3.0	0.529437	0.572434	1	1	0.735162	0.952805	0.512619

10293 rows × 22 columns

One-Hot Encoder

In [174]:
```
dummie_hot = pd.get_dummies(df_final_enc['GeoLivArea'],prefix='Area')

scaler_hot = OneHotEncoder(sparse = False ).fit(df_final_enc[['GeoLivArea']])
hot_enconded = scaler_hot.transform(df_final_enc[['GeoLivArea']])
hot_enconded = pd.DataFrame(hot_enconded, columns = dummie_hot.columns).set_index(df_final_enc.index)
```

In [175]:
```
hot_enconded
```

Out[175]:

	Area_1	Area_2	Area_3	Area_4
CustID				
1	1.0	0.0	0.0	0.0
2	0.0	0.0	0.0	1.0
3	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	1.0
5	0.0	0.0	0.0	1.0
...	...	...	...	...
10292	0.0	1.0	0.0	0.0
10293	0.0	0.0	1.0	0.0
10294	1.0	0.0	0.0	0.0
10295	0.0	1.0	0.0	0.0
10296	1.0	0.0	0.0	0.0

10293 rows × 4 columns

In [176]:
```
#(VIF): a measure of the amount of multicollinearity in regression analysis,
#which presents a scale to guide us. If this presents values above 5,
#we are facing extreme multicollinearity, which is what we should avoid.
#If it has a value lower than 5, we have a moderate level of multicollinearity and equals to 1,
#a very low level of multicollinearity
def vif(data):
    vif_data = pd.DataFrame()
    vif_data["feature"] = data.columns

    # calculating VIF for each feature
    vif_data["VIF"] = [variance_inflation_factor(data.values, i)
                       for i in range(len(data.columns))]

    return vif_data
```

In [177]:
```
#vif
vif(hot_enconded)
```

Out[177]:

	feature	VIF
0	Area_1	1.0
1	Area_2	1.0
2	Area_3	1.0
3	Area_4	1.0

In [178]:
```
df_final_enc  = pd.concat([df_final_enc, hot_enconded], axis = 1)
```

In [179]:
```
#df_final_enc.drop('GeoLivArea', axis=1, inplace=True)
```

In [180]:
```
df_final_enc
```

Out[180]:

	Age	EducDeg	MonthSal	BoxCox_Salary	Children	GeoLivArea	BoxCox_PremMotor	BoxCox_Salary%SalaryInsuranceMotor	BoxCox_PremHousehold
CustID									
1	0.265625	1.0	0.393345	0.439506	1	1	0.674537	0.928362	0.495286
2	0.062500	1.0	0.073379	0.093599	1	4	0.168151	0.514749	0.735732
3	0.453125	0.0	0.414676	0.460726	0	3	0.396153	0.871688	0.626372
4	0.281250	2.0	0.163396	0.198003	1	4	0.355713	0.749293	0.447285
5	0.406250	2.0	0.305034	0.349830	1	4	0.614866	0.899037	0.453783
...	...	...	...	...	...	...	...	...	...
10292	0.781250	3.0	0.609002	0.647790	0	2	0.702985	0.952185	0.456146
10293	0.734375	0.0	0.447526	0.493107	0	3	0.270331	0.853092	0.939385
10294	0.359375	2.0	0.551408	0.593399	1	1	0.718652	0.951311	0.551920
10295	0.343750	0.0	0.349403	0.395274	1	2	0.366201	0.848195	0.616797
10296	0.281250	3.0	0.529437	0.572434	1	1	0.735162	0.952805	0.512619

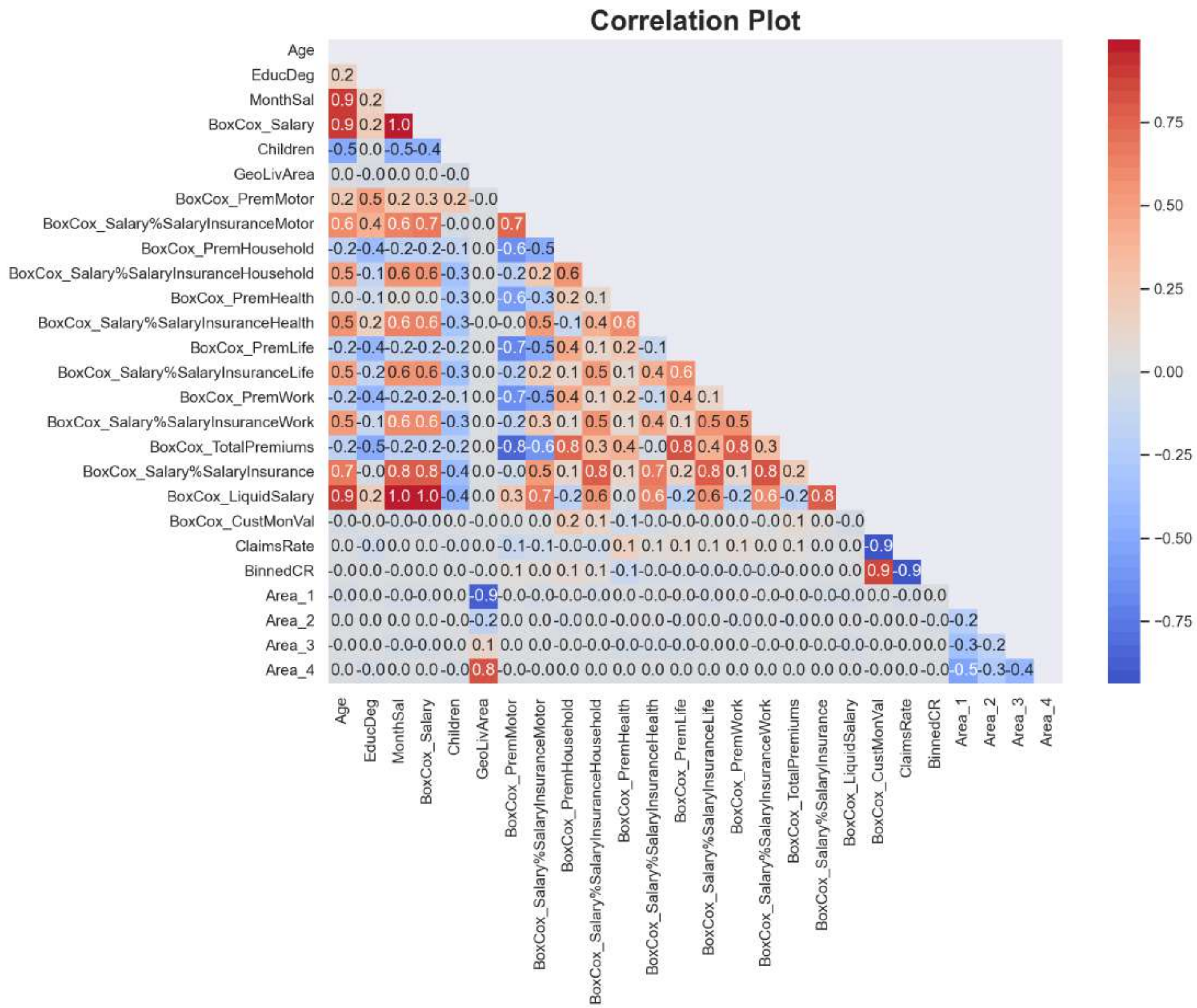
10293 rows × 26 columns

In [181]:
```
df_final_enc.describe().T
```

Out[181]:

	count	mean	std	min	25%	50%	75%	max
Age	10293.0	0.485142	0.269396	0.0	0.250000	0.484375	0.718750	1.0
EducDeg	10293.0	1.479938	0.795263	0.0	1.000000	2.000000	2.000000	3.0
MonthSal	10293.0	0.461896	0.209628	0.0	0.293089	0.462457	0.630333	1.0
BoxCox_Salary	10293.0	0.499960	0.207292	0.0	0.337448	0.507713	0.667739	1.0
Children	10293.0	0.707374	0.454990	0.0	0.000000	1.000000	1.000000	1.0
GeoLivArea	10293.0	2.709608	1.266286	1.0	1.000000	3.000000	4.000000	4.0
BoxCox_PremMotor	10293.0	0.539031	0.230855	0.0	0.367915	0.549420	0.724629	1.0
BoxCox_Salary%SalaryInsuranceMotor	10293.0	0.882724	0.113358	0.0	0.865458	0.918122	0.949330	1.0
BoxCox_PremHousehold	10293.0	0.559989	0.161970	0.0	0.455361	0.551399	0.666225	1.0
BoxCox_Salary%SalaryInsuranceHousehold	10293.0	0.849068	0.077793	0.0	0.811526	0.861420	0.902021	1.0
BoxCox_PremHealth	10293.0	0.527031	0.184042	0.0	0.395312	0.525750	0.660568	1.0
BoxCox_Salary%SalaryInsuranceHealth	10293.0	0.860421	0.088288	0.0	0.825168	0.875197	0.917089	1.0
BoxCox_PremLife	10293.0	0.531489	0.177326	0.0	0.411567	0.521102	0.656032	1.0
BoxCox_Salary%SalaryInsuranceLife	10293.0	0.817182	0.095918	0.0	0.770836	0.832625	0.883235	1.0
BoxCox_PremWork	10293.0	0.546057	0.158515	0.0	0.442238	0.536906	0.652235	1.0
BoxCox_Salary%SalaryInsuranceWork	10293.0	0.839000	0.081398	0.0	0.801216	0.852590	0.893544	1.0
BoxCox_TotalPremiums	10293.0	0.636402	0.168962	0.0	0.515696	0.643260	0.762874	1.0
BoxCox_Salary%SalaryInsurance	10293.0	0.833176	0.103398	0.0	0.782760	0.848779	0.909568	1.0
BoxCox_LiquidSalary	10293.0	0.500128	0.207287	0.0	0.337898	0.507919	0.667900	1.0
BoxCox_CustMonVal	10293.0	0.700802	0.116469	0.0	0.595066	0.703078	0.796918	1.0
ClaimsRate	10293.0	0.419663	0.196856	0.0	0.240741	0.444444	0.604938	1.0
BinnedCR	10293.0	2.516468	0.920069	1.0	2.000000	3.000000	3.000000	4.0
Area_1	10293.0	0.296124	0.456568	0.0	0.000000	0.000000	1.000000	1.0
Area_2	10293.0	0.100651	0.300881	0.0	0.000000	0.000000	0.000000	1.0
Area_3	10293.0	0.200719	0.400558	0.0	0.000000	0.000000	0.000000	1.0
Area_4	10293.0	0.402507	0.490427	0.0	0.000000	0.000000	1.000000	1.0

In [182]:
```python
df_corr = df_final_enc.corr()
corr_heatmap(df_corr)
```



In [183]:
```python
df_fs = df_final_enc.copy()
```

In [184]:
```python
df_fs.columns
```

Out[184]:
```
Index(['Age', 'EducDeg', 'MonthSal', 'BoxCox_Salary', 'Children', 'GeoLivArea',
       'BoxCox_PremMotor', 'BoxCox_Salary%SalaryInsuranceMotor',
       'BoxCox_PremHousehold', 'BoxCox_Salary%SalaryInsuranceHousehold',
       'BoxCox_PremHealth', 'BoxCox_Salary%SalaryInsuranceHealth',
       'BoxCox_PremLife', 'BoxCox_Salary%SalaryInsuranceLife',
       'BoxCox_PremWork', 'BoxCox_Salary%SalaryInsuranceWork',
       'BoxCox_TotalPremiums', 'BoxCox_Salary%SalaryInsurance',
       'BoxCox_LiquidSalary', 'BoxCox_CustMonVal', 'ClaimsRate', 'BinnedCR',
       'Area_1', 'Area_2', 'Area_3', 'Area_4'],
      dtype='object')
```

In [185]:
```
non_metric_features = ["Children","EducDeg",'GeoLivArea','BinnedCR']
metric_features = df_variables.columns.drop(non_metric_features).to_list()
metric_features
```

Out[185]:
```
['Age',
 'MonthSal',
 'BoxCox_Salary',
 'BoxCox_PremMotor',
 'BoxCox_Salary%SalaryInsuranceMotor',
 'BoxCox_PremHousehold',
 'BoxCox_Salary%SalaryInsuranceHousehold',
 'BoxCox_PremHealth',
 'BoxCox_Salary%SalaryInsuranceHealth',
 'BoxCox_PremLife',
 'BoxCox_Salary%SalaryInsuranceLife',
 'BoxCox_PremWork',
 'BoxCox_Salary%SalaryInsuranceWork',
 'BoxCox_TotalPremiums',
 'BoxCox_Salary%SalaryInsurance',
 'BoxCox_LiquidSalary',
 'BoxCox_CustMonVal',
 'ClaimsRate']
```

In [186]:
```
df_fs.describe().T
```

Out[186]:

	count	mean	std	min	25%	50%	75%	max
Age	10293.0	0.485142	0.269396	0.0	0.250000	0.484375	0.718750	1.0
EducDeg	10293.0	1.479938	0.795263	0.0	1.000000	2.000000	2.000000	3.0
MonthSal	10293.0	0.461896	0.209628	0.0	0.293089	0.462457	0.630333	1.0
BoxCox_Salary	10293.0	0.499960	0.207292	0.0	0.337448	0.507713	0.667739	1.0
Children	10293.0	0.707374	0.454990	0.0	0.000000	1.000000	1.000000	1.0
GeoLivArea	10293.0	2.709608	1.266286	1.0	1.000000	3.000000	4.000000	4.0
BoxCox_PremMotor	10293.0	0.539031	0.230855	0.0	0.367915	0.549420	0.724629	1.0
BoxCox_Salary%SalaryInsuranceMotor	10293.0	0.882724	0.113358	0.0	0.865458	0.918122	0.949330	1.0
BoxCox_PremHousehold	10293.0	0.559989	0.161970	0.0	0.455361	0.551399	0.666225	1.0
BoxCox_Salary%SalaryInsuranceHousehold	10293.0	0.849068	0.077793	0.0	0.811526	0.861420	0.902021	1.0
BoxCox_PremHealth	10293.0	0.527031	0.184042	0.0	0.395312	0.525750	0.660568	1.0
BoxCox_Salary%SalaryInsuranceHealth	10293.0	0.860421	0.088288	0.0	0.825168	0.875197	0.917089	1.0
BoxCox_PremLife	10293.0	0.531489	0.177326	0.0	0.411567	0.521102	0.656032	1.0
BoxCox_Salary%SalaryInsuranceLife	10293.0	0.817182	0.095918	0.0	0.770836	0.832625	0.883235	1.0
BoxCox_PremWork	10293.0	0.546057	0.158515	0.0	0.442238	0.536906	0.652235	1.0
BoxCox_Salary%SalaryInsuranceWork	10293.0	0.839000	0.081398	0.0	0.801216	0.852590	0.893544	1.0
BoxCox_TotalPremiums	10293.0	0.636402	0.168962	0.0	0.515696	0.643260	0.762874	1.0
BoxCox_Salary%SalaryInsurance	10293.0	0.833176	0.103398	0.0	0.782760	0.848779	0.909568	1.0
BoxCox_LiquidSalary	10293.0	0.500128	0.207287	0.0	0.337898	0.507919	0.667900	1.0
BoxCox_CustMonVal	10293.0	0.700802	0.116469	0.0	0.595066	0.703078	0.796918	1.0
ClaimsRate	10293.0	0.419663	0.196856	0.0	0.240741	0.444444	0.604938	1.0
BinnedCR	10293.0	2.516468	0.920069	1.0	2.000000	3.000000	3.000000	4.0
Area_1	10293.0	0.296124	0.456568	0.0	0.000000	0.000000	1.000000	1.0
Area_2	10293.0	0.100651	0.300881	0.0	0.000000	0.000000	0.000000	1.0
Area_3	10293.0	0.200719	0.400558	0.0	0.000000	0.000000	0.000000	1.0
Area_4	10293.0	0.402507	0.490427	0.0	0.000000	0.000000	1.000000	1.0

Scaled Data

In [187]:
```
scaler = preprocessing.MinMaxScaler(feature_range=(-1,1))
```

In [188]:
```
scaled_feat = scaler.fit_transform(df_fs[metric_features])
df_fs[metric_features] = scaled_feat
```

In [189]:
```
df_fs.describe().T
```

Out[189]:

	count	mean	std	min	25%	50%	75%	max
Age	10293.0	-0.029717	0.538792	-1.0	-0.500000	-0.031250	0.437500	1.0
EducDeg	10293.0	1.479938	0.795263	0.0	1.000000	2.000000	2.000000	3.0
MonthSal	10293.0	-0.076207	0.419255	-1.0	-0.413823	-0.075085	0.260666	1.0
BoxCox_Salary	10293.0	-0.000080	0.414584	-1.0	-0.325105	0.015427	0.335478	1.0
Children	10293.0	0.707374	0.454990	0.0	0.000000	1.000000	1.000000	1.0
GeoLivArea	10293.0	2.709608	1.266286	1.0	1.000000	3.000000	4.000000	4.0
BoxCox_PremMotor	10293.0	0.078061	0.461710	-1.0	-0.264171	0.098839	0.449259	1.0
BoxCox_Salary%SalaryInsuranceMotor	10293.0	0.765449	0.226716	-1.0	0.730917	0.836244	0.898661	1.0
BoxCox_PremHousehold	10293.0	0.119977	0.323940	-1.0	-0.089278	0.102799	0.332451	1.0
BoxCox_Salary%SalaryInsuranceHousehold	10293.0	0.698136	0.155585	-1.0	0.623052	0.722839	0.804041	1.0
BoxCox_PremHealth	10293.0	0.054063	0.368084	-1.0	-0.209377	0.051499	0.321137	1.0
BoxCox_Salary%SalaryInsuranceHealth	10293.0	0.720842	0.176577	-1.0	0.650337	0.750394	0.834177	1.0
BoxCox_PremLife	10293.0	0.062978	0.354652	-1.0	-0.176866	0.042204	0.312064	1.0
BoxCox_Salary%SalaryInsuranceLife	10293.0	0.634364	0.191836	-1.0	0.541672	0.665251	0.766470	1.0
BoxCox_PremWork	10293.0	0.092115	0.317030	-1.0	-0.115523	0.073812	0.304471	1.0
BoxCox_Salary%SalaryInsuranceWork	10293.0	0.678001	0.162797	-1.0	0.602432	0.705181	0.787087	1.0
BoxCox_TotalPremiums	10293.0	0.272804	0.337924	-1.0	0.031392	0.286519	0.525749	1.0
BoxCox_Salary%SalaryInsurance	10293.0	0.666353	0.206796	-1.0	0.565520	0.697557	0.819136	1.0
BoxCox_LiquidSalary	10293.0	0.000256	0.414575	-1.0	-0.324205	0.015838	0.335801	1.0
BoxCox_CustMonVal	10293.0	0.401604	0.232938	-1.0	0.190131	0.406156	0.593837	1.0
ClaimsRate	10293.0	-0.160675	0.393711	-1.0	-0.518519	-0.111111	0.209877	1.0
BinnedCR	10293.0	2.516468	0.920069	1.0	2.000000	3.000000	3.000000	4.0
Area_1	10293.0	0.296124	0.456568	0.0	0.000000	0.000000	1.000000	1.0
Area_2	10293.0	0.100651	0.300881	0.0	0.000000	0.000000	0.000000	1.0
Area_3	10293.0	0.200719	0.400558	0.0	0.000000	0.000000	0.000000	1.0
Area_4	10293.0	0.402507	0.490427	0.0	0.000000	0.000000	1.000000	1.0

In [190]:
```
non_metric_features = ["Children","EducDeg",'GeoLivArea']
metric_features = df_fs.columns.drop(non_metric_features).to_list()
metric_features
```

Out[190]:
```
['Age',
 'MonthSal',
 'BoxCox_Salary',
 'BoxCox_PremMotor',
 'BoxCox_Salary%SalaryInsuranceMotor',
 'BoxCox_PremHousehold',
 'BoxCox_Salary%SalaryInsuranceHousehold',
 'BoxCox_PremHealth',
 'BoxCox_Salary%SalaryInsuranceHealth',
 'BoxCox_PremLife',
 'BoxCox_Salary%SalaryInsuranceLife',
 'BoxCox_PremWork',
 'BoxCox_Salary%SalaryInsuranceWork',
 'BoxCox_TotalPremiums',
 'BoxCox_Salary%SalaryInsurance',
 'BoxCox_LiquidSalary',
 'BoxCox_CustMonVal',
 'ClaimsRate',
 'BinnedCR',
 'Area_1',
 'Area_2',
 'Area_3',
 'Area_4']
```

In [191]:
```python
metric_features = ['Age',
 'MonthSal',
 'BoxCox_Salary',
 'BoxCox_PremMotor',
 'BoxCox_Salary%SalaryInsuranceMotor',
 'BoxCox_PremHousehold',
 'BoxCox_Salary%SalaryInsuranceHousehold',
 'BoxCox_PremHealth',
 'BoxCox_Salary%SalaryInsuranceHealth',
 'BoxCox_PremLife',
 'BoxCox_Salary%SalaryInsuranceLife',
 'BoxCox_PremWork',
 'BoxCox_Salary%SalaryInsuranceWork',
 'BoxCox_TotalPremiums',
 'BoxCox_Salary%SalaryInsurance',
 'BoxCox_LiquidSalary',
 'BoxCox_CustMonVal',
 'ClaimsRate']
```

In [192]:
```python
df_fs
```

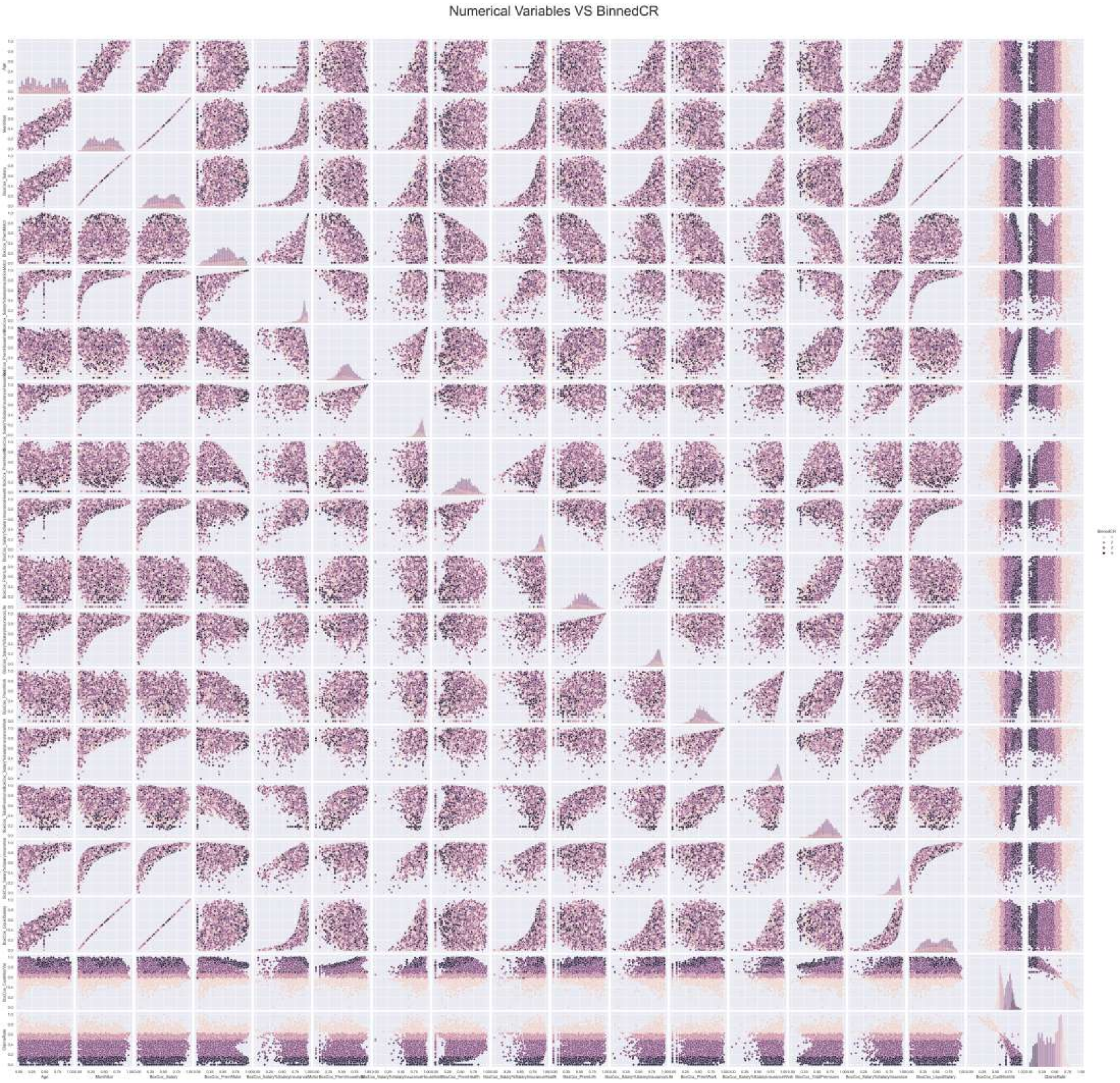
Out[192]:

CustID	Age	EducDeg	MonthSal	BoxCox_Salary	Children	GeoLivArea	BoxCox_PremMotor	BoxCox_Salary%SalaryInsuranceMotor	BoxCox_PremHousehold
1	-0.46875	1.0	-0.213311	-0.120988	1	1	0.349075	0.856723	-0.009429
2	-0.87500	1.0	-0.853242	-0.812801	1	4	-0.663698	0.029498	0.471464
3	-0.09375	0.0	-0.170648	-0.078549	0	3	-0.207693	0.743376	0.252744
4	-0.43750	2.0	-0.673208	-0.603995	1	4	-0.288574	0.498585	-0.105429
5	-0.18750	2.0	-0.389932	-0.300340	1	4	0.229733	0.798073	-0.092433
...	...	...	...	...	...	...	...	...	...
10292	0.56250	3.0	0.218003	0.295579	0	2	0.405970	0.904370	-0.087709
10293	0.46875	0.0	-0.104949	-0.013785	0	3	-0.459338	0.706183	0.878771
10294	-0.28125	2.0	0.102816	0.186797	1	1	0.437304	0.902623	0.103840
10295	-0.31250	0.0	-0.301195	-0.209452	1	2	-0.267597	0.696389	0.233593
10296	-0.43750	3.0	0.058874	0.144867	1	1	0.470325	0.905611	0.025238

10293 rows × 26 columns

Visualization (Numeric Variables VS BinnedCR)

In [193]:
```
sns.pairplot(df_final_enc[metric_features + ['BinnedCR']], diag_kind="hist", hue='BinnedCR')

plt.subplots_adjust(top=0.95)
plt.suptitle("Numerical Variables VS BinnedCR", fontsize=40)

plt.show()
```



Segmentation

Back to Index

In [194]:
```
df_fs.columns
```

Out[194]:
```
Index(['Age', 'EducDeg', 'MonthSal', 'BoxCox_Salary', 'Children', 'GeoLivArea',
       'BoxCox_PremMotor', 'BoxCox_Salary%SalaryInsuranceMotor',
       'BoxCox_PremHousehold', 'BoxCox_Salary%SalaryInsuranceHousehold',
       'BoxCox_PremHealth', 'BoxCox_Salary%SalaryInsuranceHealth',
       'BoxCox_PremLife', 'BoxCox_Salary%SalaryInsuranceLife',
       'BoxCox_PremWork', 'BoxCox_Salary%SalaryInsuranceWork',
       'BoxCox_TotalPremiums', 'BoxCox_Salary%SalaryInsurance',
       'BoxCox_LiquidSalary', 'BoxCox_CustMonVal', 'ClaimsRate', 'BinnedCR',
       'Area_1', 'Area_2', 'Area_3', 'Area_4'],
      dtype='object')
```

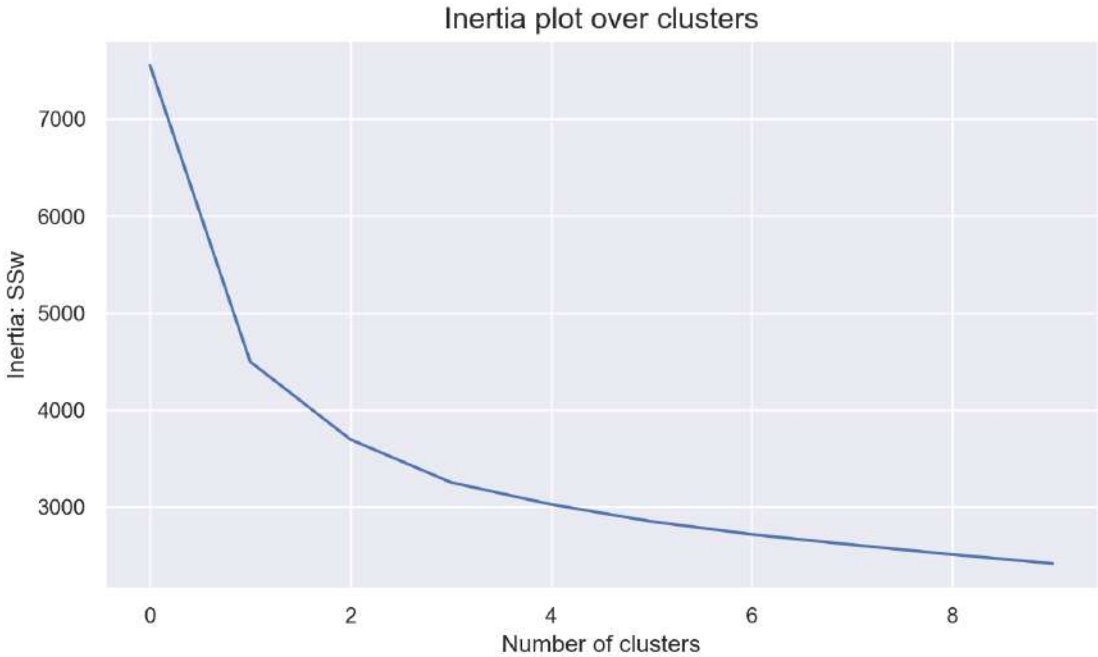
Value

In [195]:
```
value = df_fs[["BoxCox_PremHousehold","BoxCox_PremHealth","BoxCox_PremMotor","BoxCox_PremLife","BoxCox_PremWork","BoxCox_CustMonV
```

KMeans

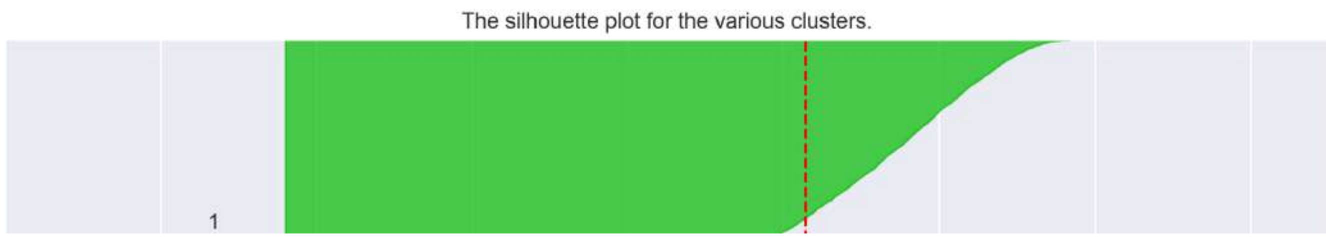
In [196]:
```
range_clusters = range(1, 11)
inertia = []
for n_clus in range_clusters:  # iterate over desired ncluster range
    kmclust = KMeans(n_clusters=n_clus, init='k-means++', n_init=15, random_state=1)
    kmclust.fit(value)
    inertia.append(kmclust.inertia_)  # save the inertia of the given cluster solution
```

In [197]:
```
# The inertia plot
plt.figure(figsize=(9,5))
plt.plot(inertia)
plt.ylabel("Inertia: SSw")
plt.xlabel("Number of clusters")
plt.title("Inertia plot over clusters", size=15)
plt.show()
```

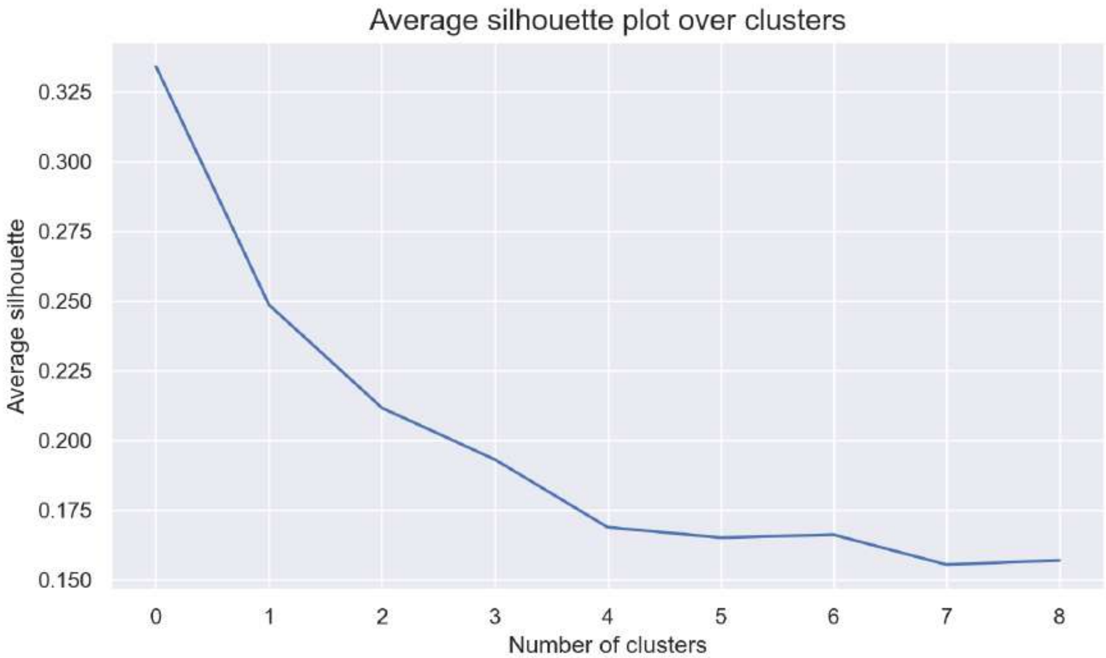


In [198]:
```python
# Adapted from:
# https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html#sphx-glr-auto-examples-cluster-plot-

# Storing average silhouette metric
avg_silhouette = []
for nclus in range_clusters:
    # Skip nclus == 1
    if nclus == 1:
        continue

    # Create a figure
    fig = plt.figure(figsize=(13, 7))

    # Initialize the KMeans object with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    kmclust = KMeans(n_clusters=nclus, init='k-means++', n_init=15, random_state=1)
    cluster_labels = kmclust.fit_predict(value)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed clusters
    silhouette_avg = silhouette_score(value, cluster_labels)
    avg_silhouette.append(silhouette_avg)
    print(f"For n_clusters = {nclus}, the average silhouette_score is : {silhouette_avg}")

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(value, cluster_labels)

    y_lower = 10
    for i in range(nclus):
        # Aggregate the silhouette scores for samples belonging to cluster i, and sort them
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels == i]
        ith_cluster_silhouette_values.sort()

        # Get y_upper to demarcate silhouette y range size
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        # Filling the silhouette
        color = cm.nipy_spectral(float(i) / nclus)
        plt.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        plt.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    plt.title("The silhouette plot for the various clusters.")
    plt.xlabel("The silhouette coefficient values")
    plt.ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    plt.axvline(x=silhouette_avg, color="red", linestyle="--")

    # The silhouette coefficient can range from -1, 1
    xmin, xmax = np.round(sample_silhouette_values.min() -0.1, 2), np.round(sample_silhouette_values.max() + 0.1, 2)
    plt.xlim([xmin, xmax])

    # The (nclus+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    plt.ylim([0, len(value) + (nclus + 1) * 10])

    plt.yticks([])  # Clear the yaxis labels / ticks
    plt.xticks(np.arange(xmin, xmax, 0.1))
```

```
For n_clusters = 2, the average silhouette_score is : 0.33408569200245564
For n_clusters = 3, the average silhouette_score is : 0.24865795259442244
For n_clusters = 4, the average silhouette_score is : 0.21169508452989955
For n_clusters = 5, the average silhouette_score is : 0.1931858775720234
For n_clusters = 6, the average silhouette_score is : 0.1688522608823906
For n_clusters = 7, the average silhouette_score is : 0.16517543900403986
For n_clusters = 8, the average silhouette_score is : 0.16621042280512885
For n_clusters = 9, the average silhouette_score is : 0.15549947632377262
For n_clusters = 10, the average silhouette_score is : 0.15702677368504345
```



In [199]:
```
# The average silhouette plot
# The inertia plot
plt.figure(figsize=(9,5))
plt.plot(avg_silhouette)
plt.ylabel("Average silhouette")
plt.xlabel("Number of clusters")
plt.title("Average silhouette plot over clusters", size=15)
plt.show()
```



In [200]:
```
# final cluster solution
number_clusters = 4
kmclust = KMeans(n_clusters=number_clusters, init='k-means++', n_init=15, random_state=1)
km_labels = kmclust.fit_predict(value)
km_labels
```

Out[200]:
```
array([0, 1, 1, ..., 0, 1, 0])
```

In [201]:
```
# Characterizing the final clusters
value['label'] = km_labels
value.groupby('label').mean()
```

Out[201]:

label	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal
0	0.069507	0.050618	0.232321	0.005148	0.040502	0.407691
1	0.491039	-0.000094	-0.494368	0.466282	0.438904	0.444372
2	-0.176129	-0.350802	0.663964	-0.281736	-0.195460	0.395995
3	0.184063	0.492790	-0.251195	0.158859	0.168627	0.364971

In [202]:
```
value['label'].value_counts()
```

Out[202]:
```
0    3440
3    2489
2    2406
1    1958
Name: label, dtype: int64
```

In [203]:
```
value
```

Out[203]:

	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal	label
CustID							
1	-0.009429	-0.021396	0.349075	0.244747	-0.019899	0.584111	0
2	0.471464	-0.173807	-0.663698	0.818614	0.539493	0.018863	1
3	0.252744	-0.132395	-0.207693	0.470097	0.512140	0.680768	1
4	-0.105429	0.717236	-0.288574	0.150285	0.101420	0.179923	3
5	-0.092433	0.154300	0.229733	-0.031379	0.209041	0.244560	0
...	...	...	...	...	...	...	...
10292	-0.087709	0.112621	0.405970	-0.176866	-0.046505	0.201349	0
10293	0.878771	-0.036971	-0.459338	-0.120570	0.535732	0.406156	1
10294	0.103840	-0.041997	0.437304	-0.124305	-0.200621	0.695109	0
10295	0.233593	0.227932	-0.267597	0.354532	0.564307	0.469815	1
10296	0.025238	-0.047033	0.470325	-0.237410	-0.071846	0.649855	0

10293 rows × 7 columns

In [204]:
```
value.drop(['label'], axis=1, inplace=True)
```

In [205]:
```
def get_ss(df):
    """Computes the sum of squares for all variables given a dataset
    """
    ss = np.sum(df.var() * (df.count() - 1))
    return ss  # return sum of sum of squares of each df variable

def r2(df, labels):
    sst = get_ss(df)
    ssw = np.sum(df.groupby(labels).apply(get_ss))
    return 1 - ssw/sst

def get_r2_scores(df, clusterer, min_k=1, max_k=8):
    """
    Loop over different values of k. To be used with sklearn clusterers.
    """
    r2_clust = {}
    for n in range(min_k, max_k):
        clust = clone(clusterer).set_params(n_clusters=n)
        labels = clust.fit_predict(df)
        r2_clust[n] = r2(df, labels)
    return r2_clust


# Set up the clusterers
kmeans = KMeans(n_clusters=4, init='k-means++', random_state=42)

hierarchical = AgglomerativeClustering(
    affinity='euclidean'
)

# Obtaining the R² scores for each cluster solution on demographic variables
r2_scores = {}
r2_scores['kmeans'] = get_r2_scores(value, kmeans)

for linkage in ['complete', 'average', 'single', 'ward']:
    r2_scores[linkage] = get_r2_scores(
        value, hierarchical.set_params(linkage=linkage)
    )

pd.DataFrame(r2_scores)
```

Out[205]:

	kmeans	complete	average	single	ward
1	0.000000	0.000000	0.000000	0.000000	0.000000
2	0.404786	0.295269	0.000532	0.000539	0.385583
3	0.510870	0.343151	0.002146	0.000848	0.463539
4	0.569367	0.432497	0.002469	0.001282	0.538010
5	0.599509	0.487894	0.003809	0.001581	0.557935
6	0.622860	0.489215	0.016667	0.001877	0.574604
7	0.640308	0.498562	0.017536	0.002279	0.591027

KMeans + Hierarchical

In [206]:
```
# final cluster solution
number_clusters = 35
kmclust = KMeans(n_clusters=number_clusters, init='k-means++', n_init=15, random_state=1)
km_labels = kmclust.fit_predict(value)
km_labels
```

Out[206]:
```
array([ 7, 11, 24, ..., 28, 33, 28])
```

In [207]:
```
mixedf = value.copy()
```

In [208]:
```
mixedf.isna().sum()
```

Out[208]:
```
BoxCox_PremHousehold    0
BoxCox_PremHealth       0
BoxCox_PremMotor        0
BoxCox_PremLife         0
BoxCox_PremWork         0
BoxCox_CustMonVal       0
dtype: int64
```

In [209]:
```
mixedf['label'] = km_labels
```

In [210]:
```
# Characterizing the final clusters
df35 = value.copy()
df35['label'] = km_labels
df35.groupby('label').mean()
```

Out[210]:

label	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal
0	-0.274451	-0.566556	0.816625	-0.370052	-0.294614	0.663350
1	0.125336	0.207244	-0.100398	0.525152	-0.139224	0.305802
2	0.689206	0.092772	-0.618273	0.655349	-0.016027	0.317340
3	-0.450085	-0.158226	0.542293	-0.139561	-0.096743	0.570480
4	0.041435	0.005334	0.348797	-0.058504	0.032237	0.205465
5	0.600846	0.305927	-0.411856	-0.116624	0.421722	0.730409
6	-0.217123	0.533360	-0.336187	0.366120	0.310322	0.234213
7	0.151591	-0.153510	0.365177	0.067240	0.092610	0.614104
8	0.354728	0.270960	0.009720	0.120648	-0.176654	0.522753
9	-0.131581	-0.374183	0.708357	-0.817177	-0.184699	0.387619
10	0.601265	-0.061861	-0.690630	0.665797	0.530060	0.749925
11	0.642861	-0.216924	-0.692568	0.635787	0.616456	0.104533
12	0.219865	0.668972	-0.345099	0.276042	0.019737	0.320722
13	0.196024	0.221893	0.144186	-0.560240	0.139669	0.433198
14	0.748787	-0.081024	-0.617332	0.096872	0.694395	0.395703
15	-0.214462	0.099384	0.116695	0.226598	0.277271	0.302696
16	0.320578	0.069296	0.037793	-0.077268	0.356688	0.449914
17	0.467002	0.473538	-0.482129	0.364108	0.252048	0.657989
18	-0.017486	-0.342546	0.630054	-0.192608	-0.076702	0.212318
19	-0.037409	-0.313538	0.624165	-0.205930	-0.105893	0.661744
20	0.034223	-0.617036	-0.776829	-0.145852	-0.049809	0.216852
21	-0.082739	0.415339	0.098483	-0.095867	-0.057352	0.272972
22	0.256885	-0.181672	0.176390	0.325532	0.151201	0.396324
23	0.336873	0.437858	-0.612461	0.498541	0.460312	0.266673
24	0.505225	-0.133201	-0.169567	0.419460	0.333505	0.695767
25	-0.301661	-0.062095	0.495757	-0.204187	-0.145013	0.144478
26	0.066222	-0.027158	0.378589	0.055944	-0.458157	0.416641
27	0.015220	-0.090754	-0.569967	0.655900	0.650666	0.327581
28	-0.020141	0.136119	0.314105	-0.140823	-0.079687	0.581025
29	0.152094	0.668879	-0.211073	-0.132564	0.134560	0.395973
30	0.317524	0.487291	-0.442775	-0.060361	0.506825	0.234856
31	0.148767	0.338374	-0.059058	0.187222	0.210048	0.421080
32	-0.368966	-0.550649	0.799202	-0.305571	-0.200310	0.121224
33	0.387846	0.096373	-0.201740	0.362263	0.368909	0.269364
34	-0.145182	-0.445870	0.753963	-0.341349	-0.597858	0.226935

In [211]:
```
meanmixed = df35.groupby(by = 'label').mean()
meanmixed
```

Out[211]:

label	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal
0	-0.274451	-0.566556	0.816625	-0.370052	-0.294614	0.663350
1	0.125336	0.207244	-0.100398	0.525152	-0.139224	0.305802
2	0.689206	0.092772	-0.618273	0.655349	-0.016027	0.317340
3	-0.450085	-0.158226	0.542293	-0.139561	-0.096743	0.570480
4	0.041435	0.005334	0.348797	-0.058504	0.032237	0.205465
5	0.600846	0.305927	-0.411856	-0.116624	0.421722	0.730409
6	-0.217123	0.533360	-0.336187	0.366120	0.310322	0.234213
7	0.151591	-0.153510	0.365177	0.067240	0.092610	0.614104
8	0.354728	0.270960	0.009720	0.120648	-0.176654	0.522753
9	-0.131581	-0.374183	0.708357	-0.817177	-0.184699	0.387619
10	0.601265	-0.061861	-0.690630	0.665797	0.530060	0.749925
11	0.642861	-0.216924	-0.692568	0.635787	0.616456	0.104533
12	0.219865	0.668972	-0.345099	0.276042	0.019737	0.320722
13	0.196024	0.221893	0.144186	-0.560240	0.139669	0.433198
14	0.748787	-0.081024	-0.617332	0.096872	0.694395	0.395703
15	-0.214462	0.099384	0.116695	0.226598	0.277271	0.302696
16	0.320578	0.069296	0.037793	-0.077268	0.356688	0.449914
17	0.467002	0.473538	-0.482129	0.364108	0.252048	0.657989
18	-0.017486	-0.342546	0.630054	-0.192608	-0.076702	0.212318
19	-0.037409	-0.313538	0.624165	-0.205930	-0.105893	0.661744
20	0.034223	-0.617036	-0.776829	-0.145852	-0.049809	0.216852
21	-0.082739	0.415339	0.098483	-0.095867	-0.057352	0.272972
22	0.256885	-0.181672	0.176390	0.325532	0.151201	0.396324
23	0.336873	0.437858	-0.612461	0.498541	0.460312	0.266673
24	0.505225	-0.133201	-0.169567	0.419460	0.333505	0.695767
25	-0.301661	-0.062095	0.495757	-0.204187	-0.145013	0.144478
26	0.066222	-0.027158	0.378589	0.055944	-0.458157	0.416641
27	0.015220	-0.090754	-0.569967	0.655900	0.650666	0.327581
28	-0.020141	0.136119	0.314105	-0.140823	-0.079687	0.581025
29	0.152094	0.668879	-0.211073	-0.132564	0.134560	0.395973
30	0.317524	0.487291	-0.442775	-0.060361	0.506825	0.234856
31	0.148767	0.338374	-0.059058	0.187222	0.210048	0.421080
32	-0.368966	-0.550649	0.799202	-0.305571	-0.200310	0.121224
33	0.387846	0.096373	-0.201740	0.362263	0.368909	0.269364
34	-0.145182	-0.445870	0.753963	-0.341349	-0.597858	0.226935

In [212]:
```
value
```

Out[212]:

CustID	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal
1	-0.009429	-0.021396	0.349075	0.244747	-0.019899	0.584111
2	0.471464	-0.173807	-0.663698	0.818614	0.539493	0.018863
3	0.252744	-0.132395	-0.207693	0.470097	0.512140	0.680768
4	-0.105429	0.717236	-0.288574	0.150285	0.101420	0.179923
5	-0.092433	0.154300	0.229733	-0.031379	0.209041	0.244560
...	...	...	...	...	...	...
10292	-0.087709	0.112621	0.405970	-0.176866	-0.046505	0.201349
10293	0.878771	-0.036971	-0.459338	-0.120570	0.535732	0.406156
10294	0.103840	-0.041997	0.437304	-0.124305	-0.200621	0.695109
10295	0.233593	0.227932	-0.267597	0.354532	0.564307	0.469815
10296	0.025238	-0.047033	0.470325	-0.237410	-0.071846	0.649855

10293 rows × 6 columns

In [213]:
```
meanmixed
```

Out[213]:

label	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal
0	-0.274451	-0.566556	0.816625	-0.370052	-0.294614	0.663350
1	0.125336	0.207244	-0.100398	0.525152	-0.139224	0.305802
2	0.689206	0.092772	-0.618273	0.655349	-0.016027	0.317340
3	-0.450085	-0.158226	0.542293	-0.139561	-0.096743	0.570480
4	0.041435	0.005334	0.348797	-0.058504	0.032237	0.205465
5	0.600846	0.305927	-0.411856	-0.116624	0.421722	0.730409
6	-0.217123	0.533360	-0.336187	0.366120	0.310322	0.234213
7	0.151591	-0.153510	0.365177	0.067240	0.092610	0.614104
8	0.354728	0.270960	0.009720	0.120648	-0.176654	0.522753
9	-0.131581	-0.374183	0.708357	-0.817177	-0.184699	0.387619
10	0.601265	-0.061861	-0.690630	0.665797	0.530060	0.749925
11	0.642861	-0.216924	-0.692568	0.635787	0.616456	0.104533
12	0.219865	0.668972	-0.345099	0.276042	0.019737	0.320722
13	0.196024	0.221893	0.144186	-0.560240	0.139669	0.433198
14	0.748787	-0.081024	-0.617332	0.096872	0.694395	0.395703
15	-0.214462	0.099384	0.116695	0.226598	0.277271	0.302696
16	0.320578	0.069296	0.037793	-0.077268	0.356688	0.449914
17	0.467002	0.473538	-0.482129	0.364108	0.252048	0.657989
18	-0.017486	-0.342546	0.630054	-0.192608	-0.076702	0.212318
19	-0.037409	-0.313538	0.624165	-0.205930	-0.105893	0.661744
20	0.034223	-0.617036	-0.776829	-0.145852	-0.049809	0.216852
21	-0.082739	0.415339	0.098483	-0.095867	-0.057352	0.272972
22	0.256885	-0.181672	0.176390	0.325532	0.151201	0.396324
23	0.336873	0.437858	-0.612461	0.498541	0.460312	0.266673
24	0.505225	-0.133201	-0.169567	0.419460	0.333505	0.695767
25	-0.301661	-0.062095	0.495757	-0.204187	-0.145013	0.144478
26	0.066222	-0.027158	0.378589	0.055944	-0.458157	0.416641
27	0.015220	-0.090754	-0.569967	0.655900	0.650666	0.327581
28	-0.020141	0.136119	0.314105	-0.140823	-0.079687	0.581025
29	0.152094	0.668879	-0.211073	-0.132564	0.134560	0.395973
30	0.317524	0.487291	-0.442775	-0.060361	0.506825	0.234856
31	0.148767	0.338374	-0.059058	0.187222	0.210048	0.421080
32	-0.368966	-0.550649	0.799202	-0.305571	-0.200310	0.121224
33	0.387846	0.096373	-0.201740	0.362263	0.368909	0.269364
34	-0.145182	-0.445870	0.753963	-0.341349	-0.597858	0.226935

In [214]:
```
df35.drop(['label'],inplace=True, axis=1)
```

In [215]:
```
df35
```

Out[215]:

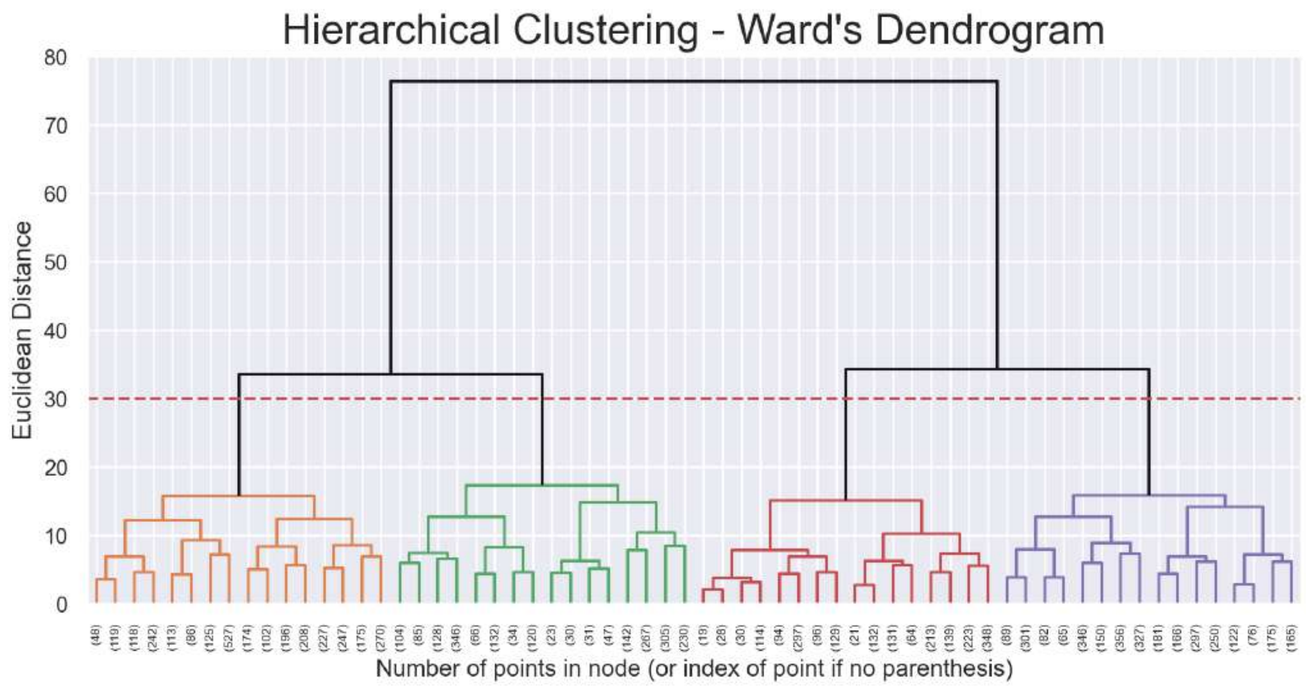
CustID	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal
1	-0.009429	-0.021396	0.349075	0.244747	-0.019899	0.584111
2	0.471464	-0.173807	-0.663698	0.818614	0.539493	0.018863
3	0.252744	-0.132395	-0.207693	0.470097	0.512140	0.680768
4	-0.105429	0.717236	-0.288574	0.150285	0.101420	0.179923
5	-0.092433	0.154300	0.229733	-0.031379	0.209041	0.244560
...	...	...	...	...	...	...
10292	-0.087709	0.112621	0.405970	-0.176866	-0.046505	0.201349
10293	0.878771	-0.036971	-0.459338	-0.120570	0.535732	0.406156
10294	0.103840	-0.041997	0.437304	-0.124305	-0.200621	0.695109
10295	0.233593	0.227932	-0.267597	0.354532	0.564307	0.469815
10296	0.025238	-0.047033	0.470325	-0.237410	-0.071846	0.649855

10293 rows × 6 columns

In [216]:
```python
linkage = 'ward'
distance = 'euclidean'
hclust = AgglomerativeClustering(linkage=linkage, affinity=distance, distance_threshold=0, n_clusters=None)
hclust.fit_predict(value)
```

Out[216]:
```
array([9625, 7587, 5417, ...,    5,    2,    0], dtype=int64)
```

In [217]:
```python
# Plotting dendrogram
counts = np.zeros(hclust.children_.shape[0])
n_samples = len(hclust.labels_)


for i, merge in enumerate(hclust.children_):
    current_count = 0
    for child_idx in merge:
        if child_idx < n_samples:
            current_count += 1
        else:
            current_count += counts[child_idx - n_samples]
    counts[i] = current_count

linkage_matrix = np.column_stack(
    [hclust.children_, hclust.distances_, counts]
).astype(float)


fig = plt.figure(figsize=(11,5))

y_threshold = 30
dendrogram(linkage_matrix, truncate_mode='level', p=5, color_threshold=y_threshold, above_threshold_color='k')
plt.hlines(y_threshold, 0, 1000, colors="r", linestyles="dashed")
plt.title(f'Hierarchical Clustering - {linkage.title()}\'s Dendrogram', fontsize=21)
plt.xlabel('Number of points in node (or index of point if no parenthesis)')
plt.ylabel(f'{distance.title()} Distance', fontsize=13)

plt.show()
```



In [218]:
```python
linkage = 'ward'
distance = 'euclidean'
num_clust = 4

final_hclust = AgglomerativeClustering(affinity = distance, linkage = linkage, n_clusters = num_clust)
final_hc_labels = final_hclust.fit_predict(value)

value['label'] = final_hc_labels
value.groupby('label').mean()
```

Out[218]:

label	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal
0	0.488424	0.007645	-0.471595	0.397778	0.403893	0.462292
1	0.058830	0.036469	0.293042	-0.077298	-0.003348	0.437457
2	-0.194363	-0.377910	0.681906	-0.284616	-0.189721	0.361180
3	0.145385	0.406779	-0.184876	0.218893	0.170903	0.349304

In [219]:
```
value
```

Out[219]:

	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal	label
CustID							
1	-0.009429	-0.021396	0.349075	0.244747	-0.019899	0.584111	1
2	0.471464	-0.173807	-0.663698	0.818614	0.539493	0.018863	0
3	0.252744	-0.132395	-0.207693	0.470097	0.512140	0.680768	0
4	-0.105429	0.717236	-0.288574	0.150285	0.101420	0.179923	3
5	-0.092433	0.154300	0.229733	-0.031379	0.209041	0.244560	1
...	...	...	...	...	...	...	...
10292	-0.087709	0.112621	0.405970	-0.176866	-0.046505	0.201349	1
10293	0.878771	-0.036971	-0.459338	-0.120570	0.535732	0.406156	0
10294	0.103840	-0.041997	0.437304	-0.124305	-0.200621	0.695109	1
10295	0.233593	0.227932	-0.267597	0.354532	0.564307	0.469815	0
10296	0.025238	-0.047033	0.470325	-0.237410	-0.071846	0.649855	1

10293 rows × 7 columns

In [220]:
```
value.drop(['label'],inplace=True, axis=1)
```

In [221]:
```
print(len(final_hc_labels))
print(len(value))
```

```
10293
10293
```

In [222]:
```
def get_r2_scores(df, clusterer, min_k=1, max_k=8):
    """
    Loop over different values of k. To be used with sklearn clusterers.
    """
    r2_clust = {}
    for n in range(min_k, max_k):
        clust = clone(clusterer).set_params(n_clusters=n)
        labels = clust.fit_predict(df)
        r2_clust[n] = r2(df, labels)
    return r2_clust
```

In [223]:
```
get_r2_scores(value, hierarchical.set_params(linkage=linkage), 2,6)
```

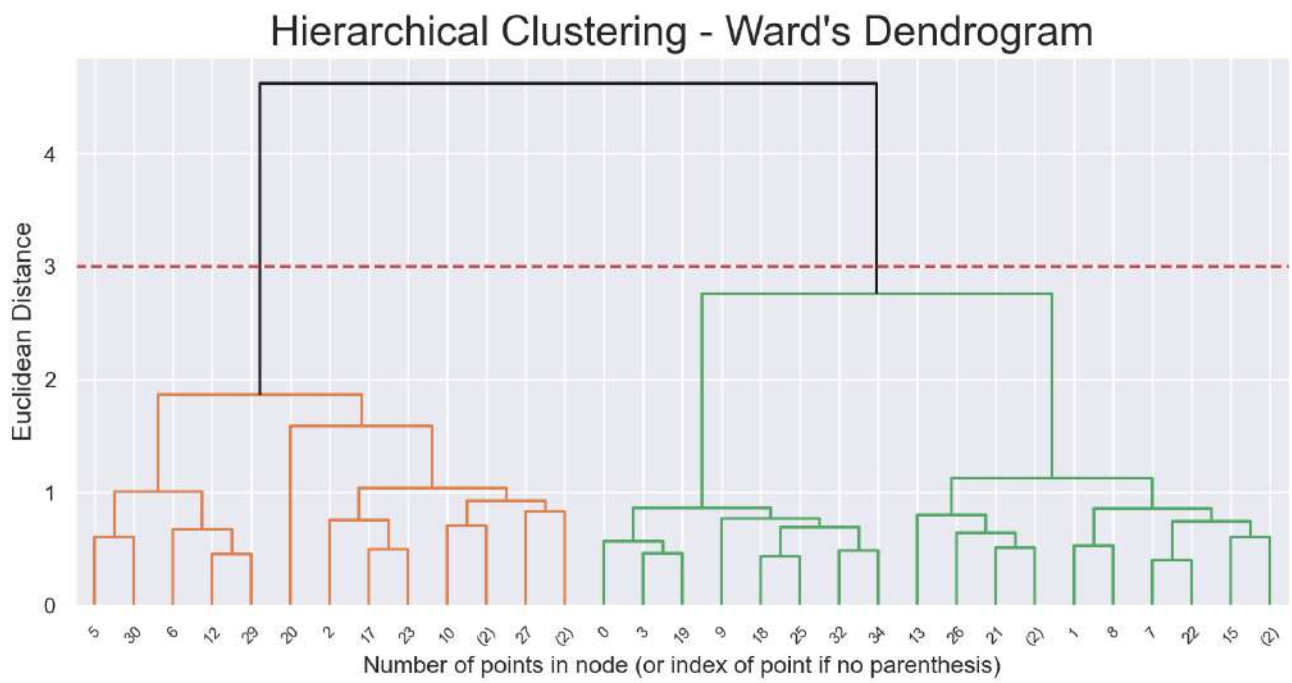
Out[223]:
```
{2: 0.3855827193470247,
 3: 0.46353869578762286,
 4: 0.5380096435123257,
 5: 0.5579351624310647}
```

In [224]:
```
linkage = 'ward'
distance = 'euclidean'
hclust = AgglomerativeClustering(linkage=linkage, affinity=distance, distance_threshold=0, n_clusters=None)
hclust.fit_predict(meanmixed)
```

Out[224]:
```
array([23, 24, 29, 30, 33, 21, 18, 34, 31, 27, 16, 19, 14, 25, 17, 22, 32,
       28, 15, 11, 20, 26,  7,  8, 12, 13,  9,  5, 10,  3,  4,  1,  6,  2,
        0], dtype=int64)
```

In [225]:

```python
# Plotting dendrogram
counts = np.zeros(hclust.children_.shape[0])
n_samples = len(hclust.labels_)


for i, merge in enumerate(hclust.children_):
    current_count = 0
    for child_idx in merge:
        if child_idx < n_samples:
            current_count += 1
        else:
            current_count += counts[child_idx - n_samples]
    counts[i] = current_count

linkage_matrix = np.column_stack(
    [hclust.children_, hclust.distances_, counts]
).astype(float)


fig = plt.figure(figsize=(11,5))

y_threshold = 3
dendrogram(linkage_matrix, truncate_mode='level', p=5, color_threshold=y_threshold, above_threshold_color='k')
plt.hlines(y_threshold, 0, 1000, colors="r", linestyles="dashed")
plt.title(f'Hierarchical Clustering - {linkage.title()}\'s Dendrogram', fontsize=21)
plt.xlabel('Number of points in node (or index of point if no parenthesis)')
plt.ylabel(f'{distance.title()} Distance', fontsize=13)

plt.show()
```



In [226]:

```python
meanmixed
```

Out[226]:

label	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal
0	-0.274451	-0.566556	0.816625	-0.370052	-0.294614	0.663350
1	0.125336	0.207244	-0.100398	0.525152	-0.139224	0.305802
2	0.689206	0.092772	-0.618273	0.655349	-0.016027	0.317340
3	-0.450085	-0.158226	0.542293	-0.139561	-0.096743	0.570480
4	0.041435	0.005334	0.348797	-0.058504	0.032237	0.205465
5	0.600846	0.305927	-0.411856	-0.116624	0.421722	0.730409
6	-0.217123	0.533360	-0.336187	0.366120	0.310322	0.234213
7	0.151591	-0.153510	0.365177	0.067240	0.092610	0.614104
8	0.354728	0.270960	0.009720	0.120648	-0.176654	0.522753
9	-0.131581	-0.374183	0.708357	-0.817177	-0.184699	0.387619
10	0.601265	-0.061861	-0.690630	0.665797	0.530060	0.749925

In [ ]:

In [227]:
```
linkage = 'ward'
distance = 'euclidean'
num_clust = 2

final_hclust = AgglomerativeClustering(affinity = distance, linkage = linkage, n_clusters = num_clust)
final_hc_labels = final_hclust.fit_predict(meanmixed)

meanmixed['label'] = final_hc_labels
meanmixed.groupby(meanmixed['label']).mean()
```

Out[227]:

label	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal
0	-0.019130	-0.070603	0.360045	-0.100040	-0.067659	0.395506
1	0.366781	0.170945	-0.478566	0.302723	0.348912	0.394527

In [228]:
```
print(len(final_hc_labels))
print(len(meanmixed))
```

```
35
35
```

In [229]:
```
def get_r2_scores(df, clusterer, min_k=1, max_k=8):
    """
    Loop over different values of k. To be used with sklearn clusterers.
    """
    r2_clust = {}
    for n in range(min_k, max_k):
        clust = clone(clusterer).set_params(n_clusters=n)
        labels = clust.fit_predict(df)
        r2_clust[n] = r2(df, labels)
    return r2_clust
```

In [230]:
```
get_r2_scores(meanmixed, hierarchical.set_params(linkage=linkage), 2,6)
```

Out[230]:
```
{2: 0.577837059737128,
 3: 0.692048407770568,
 4: 0.744198342179925,
 5: 0.7819795225473952}
```

SOM

In [231]:
```
value.columns.to_list()
```

Out[231]:
```
['BoxCox_PremHousehold',
 'BoxCox_PremHealth',
 'BoxCox_PremMotor',
 'BoxCox_PremLife',
 'BoxCox_PremWork',
 'BoxCox_CustMonVal']
```

In [232]:
```
feat = ['BoxCox_PremHousehold',
 'BoxCox_PremHealth',
 'BoxCox_PremMotor',
 'BoxCox_PremLife',
 'BoxCox_PremWork',
 'BoxCox_CustMonVal']
```

In [233]:
```python
from matplotlib.patches import RegularPolygon, Ellipse
from mpl_toolkits.axes_grid1 import make_axes_locatable
from matplotlib import cm, colorbar
from matplotlib import colors as mpl_colors

from matplotlib.lines import Line2D
from matplotlib import __version__ as mplver
from sklearn import __version__ as skv
print(skv)
from IPython.display import YouTubeVideo

from os.path import join
import pandas as pd
import numpy as np
import joblib
import sys
sys.modules['sklearn.externals.joblib'] = joblib

import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.neighbors import KNeighborsClassifier

import sompy
from sompy.visualization.mapview import View2D
from sompy.visualization.bmuhits import BmuHitsView
from sompy.visualization.hitmap import HitMapView

#!pip install ipdb
import sompy
#!pip3 install git+https://github.com/compmonks/SOMPY.git
from sompy import SOMFactory
# This som implementation does not have a random seed parameter
# We're going to set it up ourselves
# This som implementation does not have a random seed parameter
# We're going to set it up ourselves
```

```
1.0.2
```

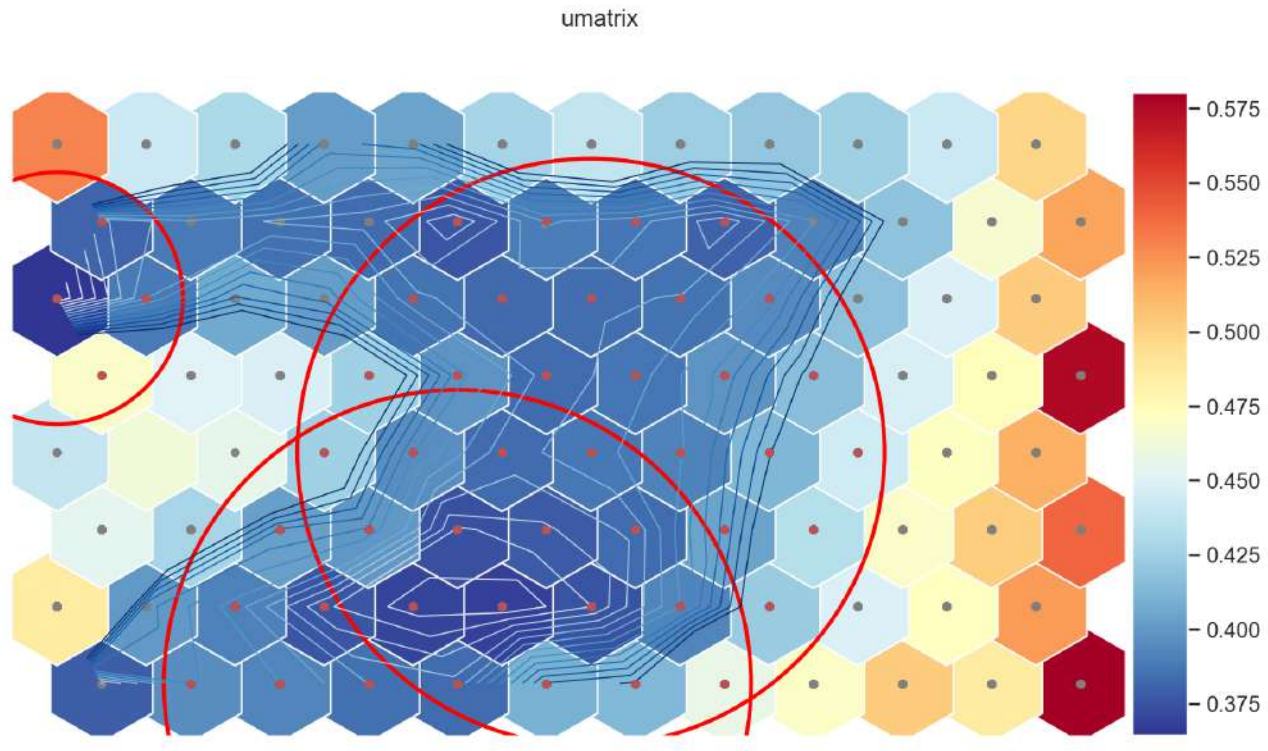
In [234]:
```python
np.random.seed(42)

sm = sompy.SOMFactory().build(
    value.values,
    mapsize=[8, 12],  # NEEDS TO BE A LIST
    initialization='random',
    neighborhood='gaussian',
    training='batch',
    lattice='hexa',
    component_names=value.columns.to_list()
)
sm.train(n_job=4, verbose='info', train_rough_len=100, train_finetune_len=100)
```

In [235]:

```python
###################################
### Visualizing Component Planes ##
###################################

def plot_component_planes(weights,
                          features,
                          M=3, N=4,
                          figsize=(20,20),
                          figlayout=(3,4),
                          title="Component Planes",
                          cmap=cm.magma
                         ):

    xx, yy = np.meshgrid(np.arange(N), np.arange(M))
    xx = xx.astype(float)
    yy = yy.astype(float)

    xx[::-2] -= 0.5

    xx = xx
    yy = yy


    weights_ = np.flipud(np.flip(weights.reshape((M,N,len(features))),axis=1))

    fig = plt.figure(figsize=figsize, constrained_layout=True)
    subfigs = fig.subfigures(figlayout[0], figlayout[1], wspace=.15)

    ## Normalize color scale to range of all values
    colornorm = mpl_colors.Normalize(vmin=np.min(weights),
                                     vmax=np.max(weights))

    for cpi, sf in zip(range(len(metric_features)), subfigs.flatten()):

        sf.suptitle(features[cpi], y=0.95, fontsize=14)

        axs = sf.subplots(1,1, )
        axs.set_aspect('equal')

        ## Normalize color scale to range of values in each component
        colornorm = mpl_colors.Normalize(vmin=np.min(weights_[:,:,cpi]),
                                         vmax=np.max(weights_[:,:,cpi]))

        # iteratively add hexagons
        for i in range(weights_.shape[0]):
            for j in range(weights_.shape[1]):
                wy = yy[(i, j)] * np.sqrt(3) / 2
                hexagon = RegularPolygon((xx[(i, j)], wy),
                                         numVertices=6,
                                         radius=.99 / np.sqrt(3),
                                         facecolor=cmap(colornorm(weights_[i, j, cpi])),
                                         alpha=1,
                                         linewidth=.5,
                                         edgecolor=cmap(colornorm(weights_[i, j, cpi]))
                                        )
                axs.add_patch(hexagon)


        ## only run this block if matplotlib >= 3.6.x
        mplv = [int(i) for i in mplver.split('.')]
        if mplv[1] >= 6:
            ## Add colorbar
            divider = make_axes_locatable(axs)

            ax_cb = divider.append_axes("right", size="7%")#, pad="2%")

            ## Create a Mappable object
            cmap_sm = plt.cm.ScalarMappable(cmap=cmap, norm=colornorm)
            cmap_sm.set_array([])

            ## Create custom colorbar
            cb1 = colorbar.Colorbar(ax_cb,
                                    orientation='vertical',
                                    alpha=1,
                                    mappable=cmap_sm
                                    )
            #cb1.ax.get_yaxis().labelpad = 16

            ## Add colorbar to plot
            sf.add_axes(ax_cb)



        ## Remove axes for hex plot
        axs.margins(.05)
        axs.axis("off")


    fig.suptitle(title, fontsize=16)
```

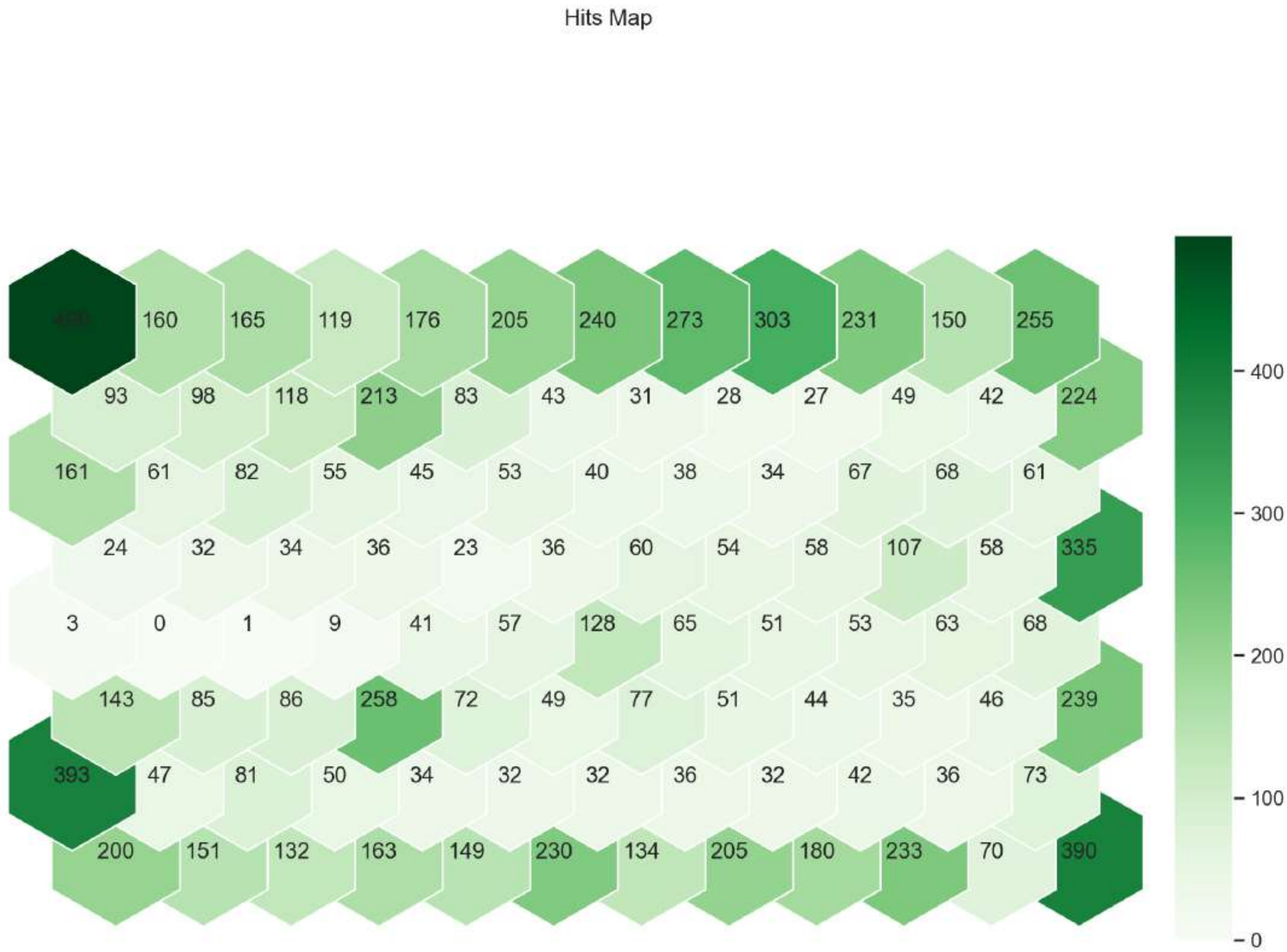
```
    plt.show()
```

In [236]:
```
# Here you have U-matrix
u = sompy.umatrix.UMatrixView(9, 9, 'umatrix', show_axis=True, text_size=8, show_text=True)

UMAT = u.show(
    sm,
    distance=2,
    row_normalized=False,
    show_data=True,
    contour=True, # Visualize isomorphic curves
    blob=True
)

np.flip(UMAT[1], axis=1) # U-matrix values - they match with the plot colors
```

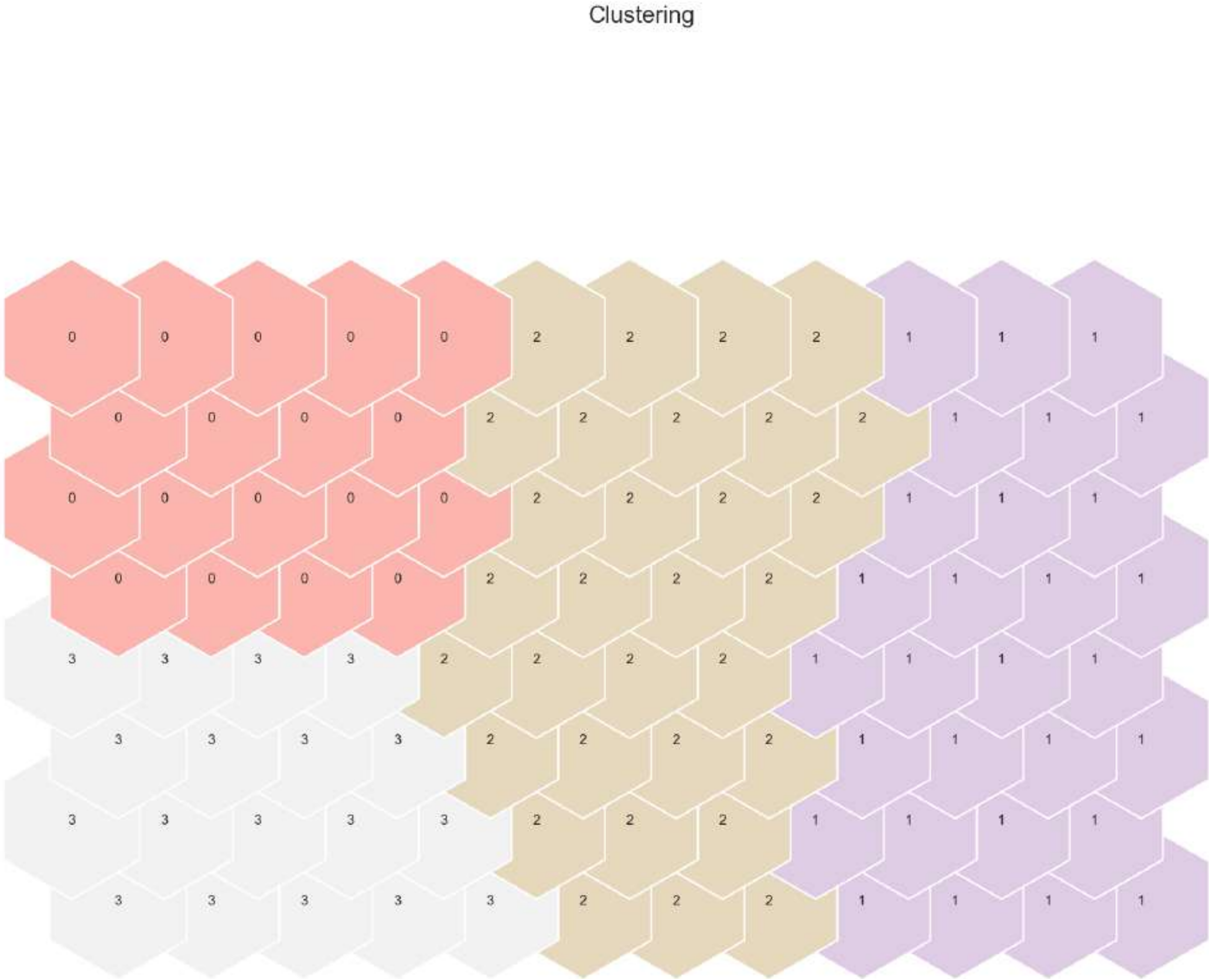


Out[236]:
```
array([[0.5289037 , 0.44265264, 0.43019581, 0.40159345, 0.40475807,
         0.42766648, 0.43907623, 0.42359448, 0.42080389, 0.42434772,
         0.44255843, 0.49825396],
        [0.38131867, 0.38866332, 0.38476521, 0.38335808, 0.37591441,
         0.38944731, 0.38665265, 0.37987226, 0.39221082, 0.41908015,
         0.46588313, 0.51866235],
        [0.36447217, 0.38882951, 0.40691594, 0.40178106, 0.38635565,
         0.3828627 , 0.38468207, 0.38575522, 0.39437753, 0.418353  ,
         0.44912226, 0.50204171],
        [0.46836075, 0.45197395, 0.44837655, 0.42546402, 0.39741434,
         0.38356595, 0.3871237 , 0.39736421, 0.41597472, 0.44737634,
         0.4733717 , 0.57548624],
        [0.43969134, 0.46299405, 0.45619947, 0.42565906, 0.39618953,
         0.38366196, 0.38763334, 0.39199538, 0.41277057, 0.44397738,
         0.47073392, 0.51367316],
        [0.45401286, 0.42801604, 0.40250618, 0.39912645, 0.37421609,
         0.37788017, 0.38315363, 0.40409899, 0.43382786, 0.46847269,
         0.5016007 , 0.53940908],
        [0.48556002, 0.40063754, 0.39080561, 0.37562774, 0.36982443,
         0.36810817, 0.37492394, 0.39250858, 0.42324639, 0.44945977,
         0.47174944, 0.52141809],
        [0.37801368, 0.39628274, 0.39260876, 0.38540771, 0.38312112,
         0.41031128, 0.41408492, 0.45727696, 0.46920811, 0.50193216,
         0.48502529, 0.58025514]])
```

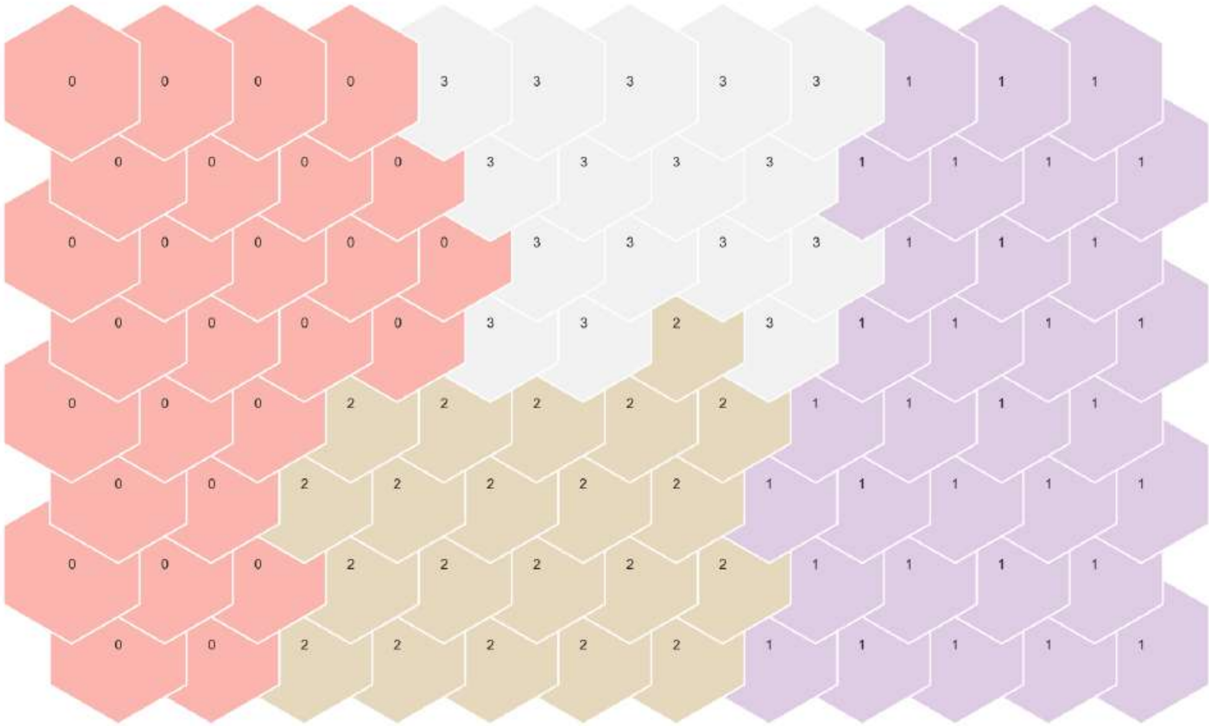
In [237]:
```
vhts  = BmuHitsView(12,12,"Hits Map")
vhts.show(sm, anotate=True, onlyzeros=False, labelsize=12, cmap="Greens")
plt.show()
```



In [238]:
```python
# Perform K-Means clustering on top of the 2500 untis (sm.get_node_vectors() output)
kmeans = KMeans(n_clusters=4, init='k-means++', n_init=20, random_state=42)
nodeclus_labels = kmeans.fit_predict(sm.codebook.matrix)
sm.cluster_labels = nodeclus_labels  # setting the cluster labels of sompy

hits = HitMapView(12, 12,"Clustering", text_size=10)
hits.show(sm, anotate=True, onlyzeros=False, labelsize=7, cmap="Pastel1")

plt.show()
```



In [239]:
```
hierclust = AgglomerativeClustering(n_clusters=4, linkage='ward')
nodeclus_labels = hierclust.fit_predict(sm.codebook.matrix)
sm.cluster_labels = nodeclus_labels  # setting the cluster labels of sompy

hits  = HitMapView(12, 12,"Clustering",text_size=10)
hits.show(sm, anotate=True, onlyzeros=False, labelsize=7, cmap="Pastel1")

plt.show()
```



In [240]:
```
# Check the nodes and and respective clusters
nodes = sm.codebook.matrix

df_nodes = pd.DataFrame(nodes, columns=value.columns.to_list())
df_nodes['label'] = nodeclus_labels
df_nodes
```

Out[240]:

	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal	label
0	-0.164673	0.803837	-0.953495	0.848079	0.877204	-0.714971	1
1	0.097071	1.061957	-0.921455	0.620378	0.646680	-0.571419	1
2	0.196746	1.270160	-0.835661	0.413582	0.315924	-0.384275	1
3	0.121020	1.316530	-0.648728	0.110910	0.069309	-0.210645	3
4	-0.105687	1.187219	-0.416457	-0.035615	-0.201486	-0.281067	3
...	...	...	...	...	...	...	...
91	0.057553	-0.282102	0.428324	-0.061232	-0.224177	0.749282	2
92	-0.149428	-0.390664	0.630695	-0.329750	-0.311570	0.839872	2
93	-0.344645	-0.560936	0.830564	-0.544899	-0.505946	0.887543	2
94	-0.561428	-0.805301	1.037535	-0.713608	-0.699027	0.945762	0
95	-0.753787	-1.074213	1.227293	-0.890089	-0.843587	0.999910	0

96 rows × 7 columns

In [241]:
```python
# Obtaining SOM's BMUs labels
bmus_map = sm.find_bmu(value)[0]  # get bmus for each observation in df

df_bmus = pd.DataFrame(
    np.concatenate((value, np.expand_dims(bmus_map,1)), axis=1),
    index=df.index, columns=np.append(value.columns,"BMU")
)
df_bmus
```

Out[241]:

CustID	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal	BMU
1	-0.009429	-0.021396	0.349075	0.244747	-0.019899	0.584111	78.0
2	0.471464	-0.173807	-0.663698	0.818614	0.539493	0.018863	50.0
3	0.252744	-0.132395	-0.207693	0.470097	0.512140	0.680768	76.0
4	-0.105429	0.717236	-0.288574	0.150285	0.101420	0.179923	41.0
5	-0.092433	0.154300	0.229733	-0.031379	0.209041	0.244560	53.0
...	...	...	...	...	...	...	...
10292	-0.087709	0.112621	0.405970	-0.176866	-0.046505	0.201349	54.0
10293	0.878771	-0.036971	-0.459338	-0.120570	0.535732	0.406156	88.0
10294	0.103840	-0.041997	0.437304	-0.124305	-0.200621	0.695109	78.0
10295	0.233593	0.227932	-0.267597	0.354532	0.564307	0.469815	64.0
10296	0.025238	-0.047033	0.470325	-0.237410	-0.071846	0.649855	78.0

10293 rows × 7 columns

In [242]:
```python
# Get cluster labels for each observation
df_final = df_bmus.merge(df_nodes['label'], 'left', left_on="BMU", right_index=True)
df_final
```

Out[242]:

CustID	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal	BMU	label
1	-0.009429	-0.021396	0.349075	0.244747	-0.019899	0.584111	78.0	2
2	0.471464	-0.173807	-0.663698	0.818614	0.539493	0.018863	50.0	1
3	0.252744	-0.132395	-0.207693	0.470097	0.512140	0.680768	76.0	2
4	-0.105429	0.717236	-0.288574	0.150285	0.101420	0.179923	41.0	2
5	-0.092433	0.154300	0.229733	-0.031379	0.209041	0.244560	53.0	2
...	...	...	...	...	...	...	...	...
10292	-0.087709	0.112621	0.405970	-0.176866	-0.046505	0.201349	54.0	2
10293	0.878771	-0.036971	-0.459338	-0.120570	0.535732	0.406156	88.0	1
10294	0.103840	-0.041997	0.437304	-0.124305	-0.200621	0.695109	78.0	2
10295	0.233593	0.227932	-0.267597	0.354532	0.564307	0.469815	64.0	1
10296	0.025238	-0.047033	0.470325	-0.237410	-0.071846	0.649855	78.0	2

10293 rows × 8 columns

In [243]:
```python
# Characterizing the final clusters
df_final.drop(columns='BMU').groupby('label').mean()
```

Out[243]:

label	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal
0	-0.338493	-0.541080	0.798561	-0.395791	-0.277508	0.154782
1	0.505458	0.170854	-0.544930	0.423567	0.417434	0.430204
2	0.044161	0.022236	0.227563	-0.012793	0.025744	0.417503
3	0.043305	0.640179	-0.302004	0.114085	0.057165	0.182111

In [ ]:

In [244]:
```python
# using R²
def get_ss(df):
    ss = np.sum(df.var() * (df.count() - 1))
    return ss  # return sum of sum of squares of each df variable

sst = get_ss(value)  # get total sum of squares
ssw_labels = df_final[feat + ["label"]].groupby(by='label').apply(get_ss)  # compute ssw for each cluster labels
ssb = sst - np.sum(ssw_labels)  # remember: SST = SSW + SSB
r2 = ssb / sst
r2
```

Out[244]: 0.3492781043146812

Demographic

In [245]:
```
demogra = df_fs[["EducDeg","Age","BoxCox_Salary","Children",'Area_1', 'Area_2', 'Area_3', 'Area_4','BinnedCR','BoxCox_Salary%Sala
```

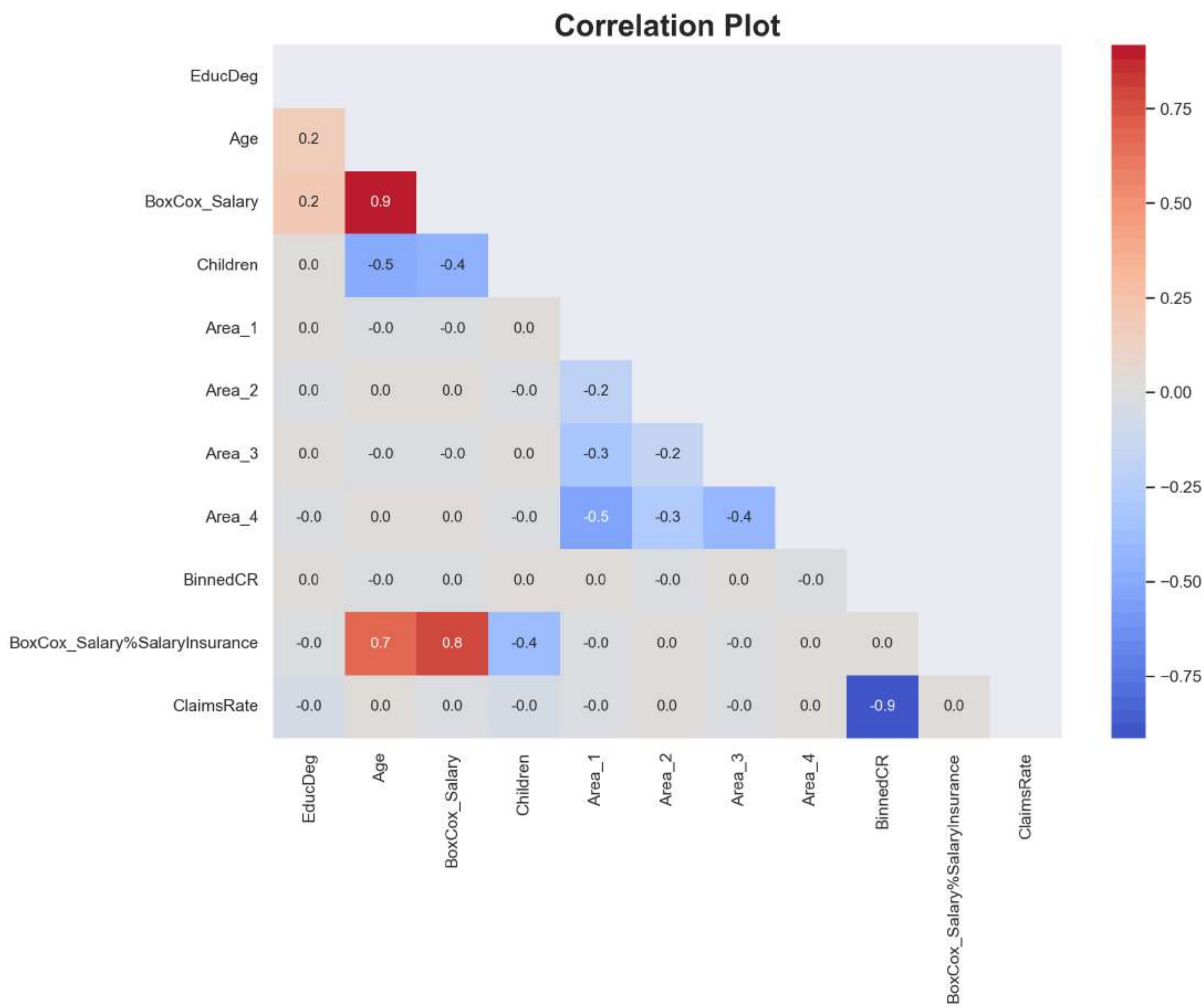
In [246]:
```
demogra
```

Out[246]:

	EducDeg	Age	BoxCox_Salary	Children	Area_1	Area_2	Area_3	Area_4	BinnedCR	BoxCox_Salary%SalaryInsurance	ClaimsRate
CustID											
1	1.0	-0.46875	-0.120988	1	1.0	0.0	0.0	0.0	3	0.676132	-0.518519
2	1.0	-0.87500	-0.812801	1	0.0	0.0	0.0	1.0	1	0.371473	0.382716
3	0.0	-0.09375	-0.078549	0	0.0	0.0	1.0	0.0	3	0.808431	-0.654321
4	2.0	-0.43750	-0.603995	1	0.0	0.0	0.0	1.0	2	0.309289	0.222222
5	2.0	-0.18750	-0.300340	1	0.0	0.0	0.0	1.0	2	0.565863	0.111111
...	...	...	...	...	...	...	...	...	...	...	...
10292	3.0	0.56250	0.295579	0	0.0	1.0	0.0	0.0	2	0.756852	0.185185
10293	0.0	0.46875	-0.013785	0	0.0	0.0	1.0	0.0	4	0.821071	-1.000000
10294	2.0	-0.28125	0.186797	1	1.0	0.0	0.0	0.0	4	0.729826	-0.740741
10295	0.0	-0.31250	-0.209452	1	0.0	1.0	0.0	0.0	3	0.767775	-0.197531
10296	3.0	-0.43750	0.144867	1	1.0	0.0	0.0	0.0	3	0.710838	-0.666667

10293 rows × 11 columns

In [247]:
```
demogra_corr = demogra.corr()
corr_heatmap(demogra_corr)
```

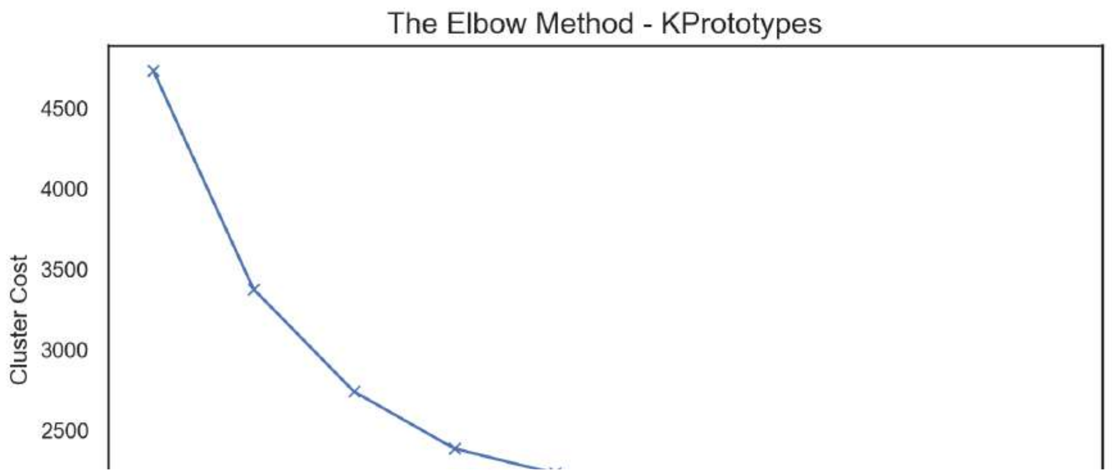


K-Prototype

In [248]:
```python
demographic=demogra[['Children', 'EducDeg', 'BoxCox_Salary', 'BinnedCR']]
categorical_columns = [0,1,3]
```

In [249]:
```python
# Elbow plot
cost_elbow = []
for n_clus in range(1,11):
    kproto = KPrototypes(n_clusters= n_clus, init='Huang',n_jobs = 1)
    clusters = kproto.fit_predict(demographic, categorical=categorical_columns)
    cost_elbow.append(kproto.cost_)
```

In [250]:
```python
# Plot elbow plot
sns.set_style("white")
range_K=range(1,11)
plt.figure(figsize=(9,5))
plt.plot(range_K, cost_elbow, 'bx-')
pd.Series(cost_elbow,index=range_K)
plt.xlabel('Number of clusters')
plt.ylabel('Cluster Cost')
plt.title('The Elbow Method - KPrototypes', size=15)

plt.show()
```



In [251]:
```python
kproto_3 = KPrototypes(n_clusters= 3, init='Huang', n_jobs = 1, random_state=42)
clusters_3 = kproto_3.fit_predict(demographic, categorical=categorical_columns)

pd.Series(clusters_3).value_counts()
```

Out[251]:
```
2    3685
1    3660
0    2948
dtype: int64
```

In [252]:
```python
clusters_3_df=pd.DataFrame(clusters_3+1,index=demographic.index).rename(columns={0:'label'})
clusters_3_df
```

Out[252]:

	label
CustID	
1	3
2	3
3	1
4	2
5	2
...	...
10292	1
10293	2
10294	2
10295	3
10296	2

10293 rows × 1 columns

In [253]:
```
#social_labels.drop(['social_labels'], axis =1 , inplace=True)
demographic['label'] = clusters_3_df
demographic.groupby('label').mean()
```

Out[253]:

label	Children	EducDeg	BoxCox_Salary	BinnedCR
1	0.196744	1.570217	0.465176	2.570896
2	0.945355	1.757377	0.019244	2.343716
3	0.879512	1.132157	-0.391477	2.644505

In [254]:
```
demographic.drop(['label'], axis=1,inplace=True)
```

In [255]:
```
demographic
```

Out[255]:

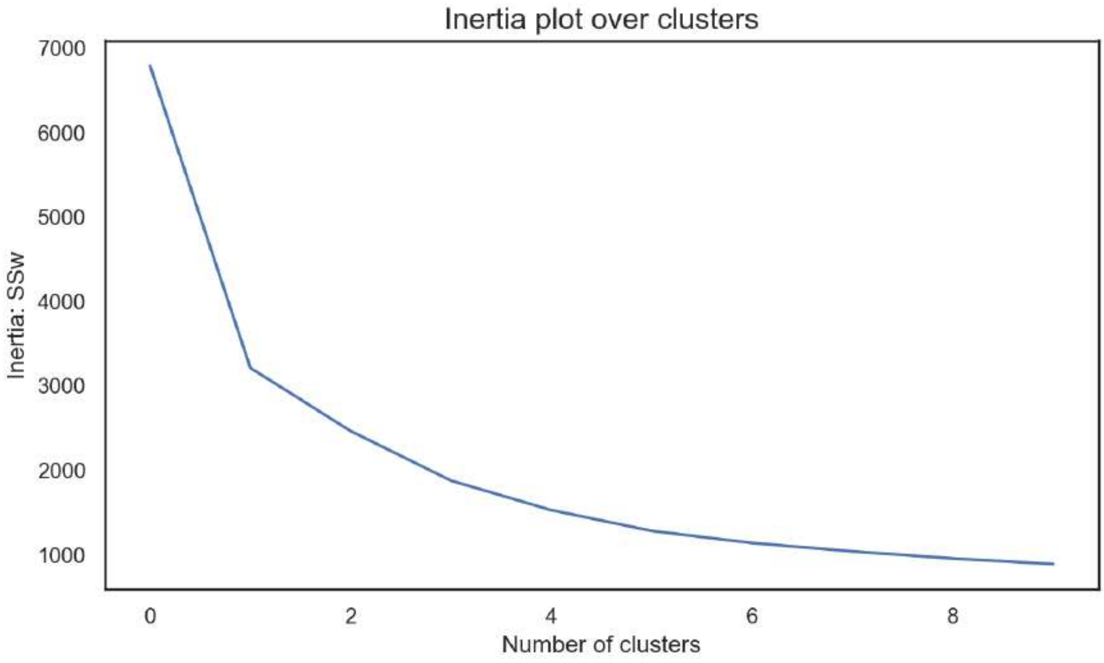
CustID	Children	EducDeg	BoxCox_Salary	BinnedCR
1	1	1.0	-0.120988	3
2	1	1.0	-0.812801	1
3	0	0.0	-0.078549	3
4	1	2.0	-0.603995	2
5	1	2.0	-0.300340	2
...	...	...	...	...
10292	0	3.0	0.295579	2
10293	0	0.0	-0.013785	4
10294	1	2.0	0.186797	4
10295	1	0.0	-0.209452	3
10296	1	3.0	0.144867	3

10293 rows × 4 columns

KMeans

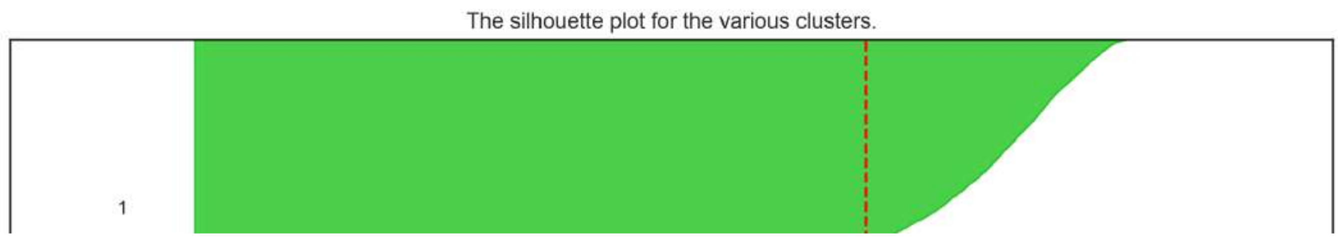
In [256]:
```
demographic_kmeans=demogra[['Age', 'ClaimsRate', 'BoxCox_Salary','BoxCox_Salary%SalaryInsurance']]
```

In [257]:
```
range_clusters = range(1, 11)
inertia = []
for n_clus in range_clusters:  # iterate over desired ncluster range
    kmclust = KMeans(n_clusters=n_clus, init='k-means++', n_init=15, random_state=1)
    kmclust.fit(demographic_kmeans)
    inertia.append(kmclust.inertia_)  # save the inertia of the given cluster solution
# The inertia plot
plt.figure(figsize=(9,5))
plt.plot(inertia)
plt.ylabel("Inertia: SSw")
plt.xlabel("Number of clusters")
plt.title("Inertia plot over clusters", size=15)
plt.show()
```

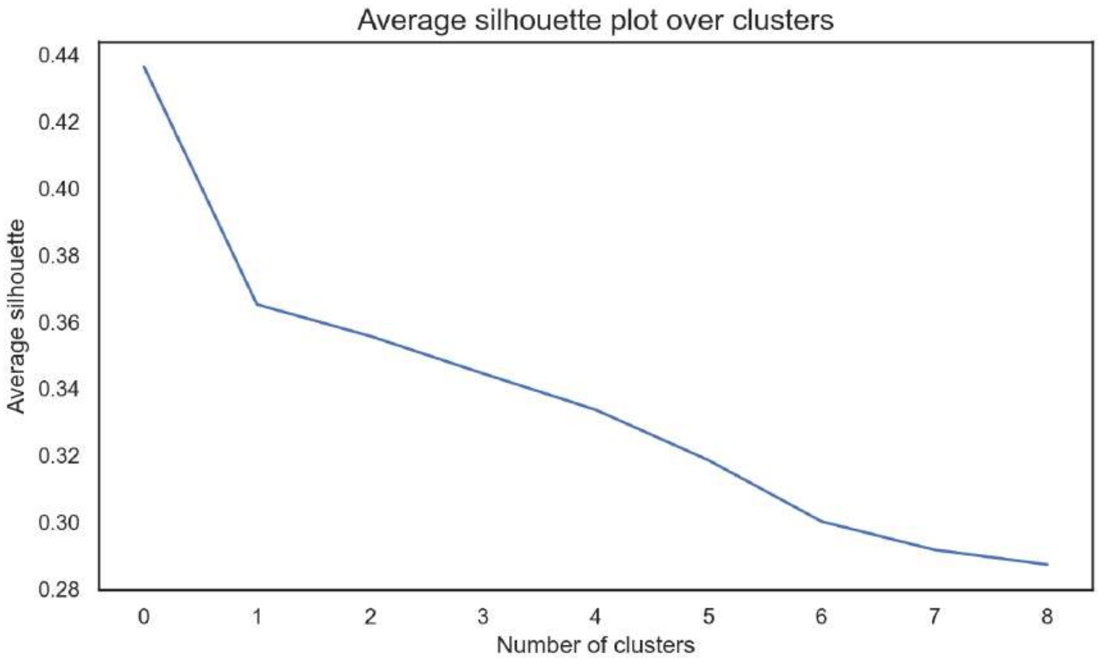


In [258]:
```python
# Adapted from:
# https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html#sphx-glr-auto-examples-cluster-plot-

# Storing average silhouette metric
avg_silhouette = []
for nclus in range_clusters:
    # Skip nclus == 1
    if nclus == 1:
        continue

    # Create a figure
    fig = plt.figure(figsize=(13, 7))

    # Initialize the KMeans object with n_clusters demographic_kmeans and a random generator
    # seed of 10 for reproducibility.
    kmclust = KMeans(n_clusters=nclus, init='k-means++', n_init=15, random_state=1)
    cluster_labels = kmclust.fit_predict(demographic_kmeans)

    # The silhouette_score gives the average demographic_kmeans for all the samples.
    # This gives a perspective into the density and separation of the formed clusters
    silhouette_avg = silhouette_score(demographic_kmeans, cluster_labels)
    avg_silhouette.append(silhouette_avg)
    print(f"For n_clusters = {nclus}, the average silhouette_score is : {silhouette_avg}")

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(demographic_kmeans, cluster_labels)

    y_lower = 10
    for i in range(nclus):
        # Aggregate the silhouette scores for samples belonging to cluster i, and sort them
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels == i]
        ith_cluster_silhouette_values.sort()

        # Get y_upper to demarcate silhouette y range size
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        # Filling the silhouette
        color = cm.nipy_spectral(float(i) / nclus)
        plt.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        plt.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    plt.title("The silhouette plot for the various clusters.")
    plt.xlabel("The silhouette coefficient values")
    plt.ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    plt.axvline(x=silhouette_avg, color="red", linestyle="--")

    # The silhouette coefficient can range from -1, 1
    xmin, xmax = np.round(sample_silhouette_values.min() -0.1, 2), np.round(sample_silhouette_values.max() + 0.1, 2)
    plt.xlim([xmin, xmax])

    # The (nclus+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    plt.ylim([0, len(demographic_kmeans) + (nclus + 1) * 10])

    plt.yticks([])  # Clear the yaxis labels / ticks
    plt.xticks(np.arange(xmin, xmax, 0.1))
```

```
For n_clusters = 2, the average silhouette_score is : 0.4364282687295146
For n_clusters = 3, the average silhouette_score is : 0.36523875472958733
For n_clusters = 4, the average silhouette_score is : 0.3557659679552305
For n_clusters = 5, the average silhouette_score is : 0.34455230166315365
For n_clusters = 6, the average silhouette_score is : 0.3336863111240779
For n_clusters = 7, the average silhouette_score is : 0.3185754278724673
For n_clusters = 8, the average silhouette_score is : 0.30023617772514627
For n_clusters = 9, the average silhouette_score is : 0.29178175001675816
For n_clusters = 10, the average silhouette_score is : 0.2873506640108026
```



In [259]:
```python
# The average silhouette plot
# The inertia plot
plt.figure(figsize=(9,5))
plt.plot(avg_silhouette)
plt.ylabel("Average silhouette")
plt.xlabel("Number of clusters")
plt.title("Average silhouette plot over clusters", size=15)
plt.show()
```



In [260]:
```python
# final cluster solution
number_clusters = 3
kmclust = KMeans(n_clusters=number_clusters, init='k-means++', n_init=15, random_state=1)
km_labels = kmclust.fit_predict(demographic_kmeans)

pd.Series(km_labels).value_counts()
```

Out[260]:
```
1    4163
2    4031
0    2099
dtype: int64
```

In [261]:
```python
# Characterizing the final clusters
demographic_kmeans['label'] = km_labels
demographic_kmeans.groupby('label').mean()
```

Out[261]:

label	Age	ClaimsRate	BoxCox_Salary	BoxCox_Salary%SalaryInsurance
0	0.027781	-0.650127	0.055547	0.677869
1	-0.560383	-0.059246	-0.396989	0.516461
2	0.488387	-0.010560	0.380861	0.815157

In [262]:
```python
demographic_kmeans
```

Out[262]:

CustID	Age	ClaimsRate	BoxCox_Salary	BoxCox_Salary%SalaryInsurance	label
1	-0.46875	-0.518519	-0.120988	0.676132	0
2	-0.87500	0.382716	-0.812801	0.371473	1
3	-0.09375	-0.654321	-0.078549	0.808431	0
4	-0.43750	0.222222	-0.603995	0.309289	1
5	-0.18750	0.111111	-0.300340	0.565863	1
...	...	...	...	...	...
10292	0.56250	0.185185	0.295579	0.756852	2
10293	0.46875	-1.000000	-0.013785	0.821071	0
10294	-0.28125	-0.740741	0.186797	0.729826	0
10295	-0.31250	-0.197531	-0.209452	0.767775	1
10296	-0.43750	-0.666667	0.144867	0.710838	0

10293 rows × 5 columns

In [263]:
```
demographic_kmeans.drop('label',axis=1,inplace=True)
```

KMeans + Hierarchical

In [264]:
```
# final cluster solution
number_clusters = 35
kmclust = KMeans(n_clusters=number_clusters, init='k-means++', n_init=15, random_state=1)
km_labels = kmclust.fit_predict(demographic_kmeans)
km_labels
```

Out[264]:
```
array([14, 23,  0, ...,  0, 27,  0])
```

In [265]:
```
mixedf = demographic_kmeans.copy()
```

In [266]:
```
mixedf['label'] = km_labels
```

In [267]:
```
# Characterizing the final clusters
df35 = demographic_kmeans.copy()
df35['label'] = km_labels
df35.groupby('label').mean()
```

Out[267]:

label	Age	ClaimsRate	BoxCox_Salary	BoxCox_Salary%SalaryInsurance
0	-0.164569	-0.569460	-0.000423	0.677952
1	0.674260	0.276348	0.474091	0.850652
2	-0.774359	-0.163484	-0.552283	0.528701
3	-0.115132	0.243165	-0.247396	0.505477
4	0.166221	-0.007974	0.104513	0.767506
5	0.801941	-0.257110	0.656988	0.889027
6	0.577427	-0.365715	0.434947	0.858920
7	-0.814312	-0.641260	-0.833854	-0.112467
8	-0.585653	-0.132304	-0.243976	0.672245
9	-0.349130	0.216282	-0.137117	0.639704
10	0.159824	-0.755403	0.217323	0.734051
11	0.666972	-0.041671	0.473596	0.863740
12	-0.850922	0.323872	-0.522317	0.617412
13	-0.797945	0.162185	-0.897091	-0.293265
14	-0.465909	-0.579529	-0.173802	0.648241
15	-0.350953	-0.268025	-0.365261	0.558415
16	0.389596	-0.149735	0.292286	0.834164
17	-0.744591	-0.208452	-0.711563	0.210704
18	0.456871	-0.664717	0.366301	0.806863
19	-0.444433	0.235163	-0.433455	0.420510
20	0.481696	0.210053	0.223618	0.770849
21	-0.647211	0.167024	-0.321421	0.634316
22	-0.846721	-0.671455	-0.618912	0.497413
23	-0.810409	0.224471	-0.697086	0.309320
24	0.771844	-0.610595	0.583832	0.861370
25	0.161643	0.269795	0.172737	0.716821
26	0.816935	0.195141	0.676448	0.877548
27	-0.194473	-0.137639	-0.057362	0.722454
28	0.023821	-0.802118	0.000066	0.598501
29	-0.755743	-0.497309	-0.399227	0.650161
30	-0.046524	0.246257	0.022338	0.641460
31	0.258690	-0.436364	0.181061	0.786636
32	0.367863	0.209915	0.439345	0.842868
33	-0.220433	-0.726353	-0.259103	0.493749
34	-0.512876	-0.594023	-0.456807	0.395205

In [268]:
```
meanmixed = df35.groupby(by = 'label').mean()
meanmixed
```

Out[268]:

label	Age	ClaimsRate	BoxCox_Salary	BoxCox_Salary%SalaryInsurance
0	-0.164569	-0.569460	-0.000423	0.677952
1	0.674260	0.276348	0.474091	0.850652
2	-0.774359	-0.163484	-0.552283	0.528701
3	-0.115132	0.243165	-0.247396	0.505477
4	0.166221	-0.007974	0.104513	0.767506
5	0.801941	-0.257110	0.656988	0.889027
6	0.577427	-0.365715	0.434947	0.858920
7	-0.814312	-0.641260	-0.833854	-0.112467
8	-0.585653	-0.132304	-0.243976	0.672245
9	-0.349130	0.216282	-0.137117	0.639704
10	0.159824	-0.755403	0.217323	0.734051
11	0.666972	-0.041671	0.473596	0.863740
12	-0.850922	0.323872	-0.522317	0.617412
13	-0.797945	0.162185	-0.897091	-0.293265
14	-0.465909	-0.579529	-0.173802	0.648241
15	-0.350953	-0.268025	-0.365261	0.558415
16	0.389596	-0.149735	0.292286	0.834164
17	-0.744591	-0.208452	-0.711563	0.210704
18	0.456871	-0.664717	0.366301	0.806863
19	-0.444433	0.235163	-0.433455	0.420510
20	0.481696	0.210053	0.223618	0.770849
21	-0.647211	0.167024	-0.321421	0.634316
22	-0.846721	-0.671455	-0.618912	0.497413
23	-0.810409	0.224471	-0.697086	0.309320
24	0.771844	-0.610595	0.583832	0.861370
25	0.161643	0.269795	0.172737	0.716821
26	0.816935	0.195141	0.676448	0.877548
27	-0.194473	-0.137639	-0.057362	0.722454
28	0.023821	-0.802118	0.000066	0.598501
29	-0.755743	-0.497309	-0.399227	0.650161
30	-0.046524	0.246257	0.022338	0.641460
31	0.258690	-0.436364	0.181061	0.786636
32	0.367863	0.209915	0.439345	0.842868
33	-0.220433	-0.726353	-0.259103	0.493749
34	-0.512876	-0.594023	-0.456807	0.395205

In [269]:
```
df35.drop(['label'],inplace=True, axis=1)
```

In [270]:
```
df35
```

Out[270]:

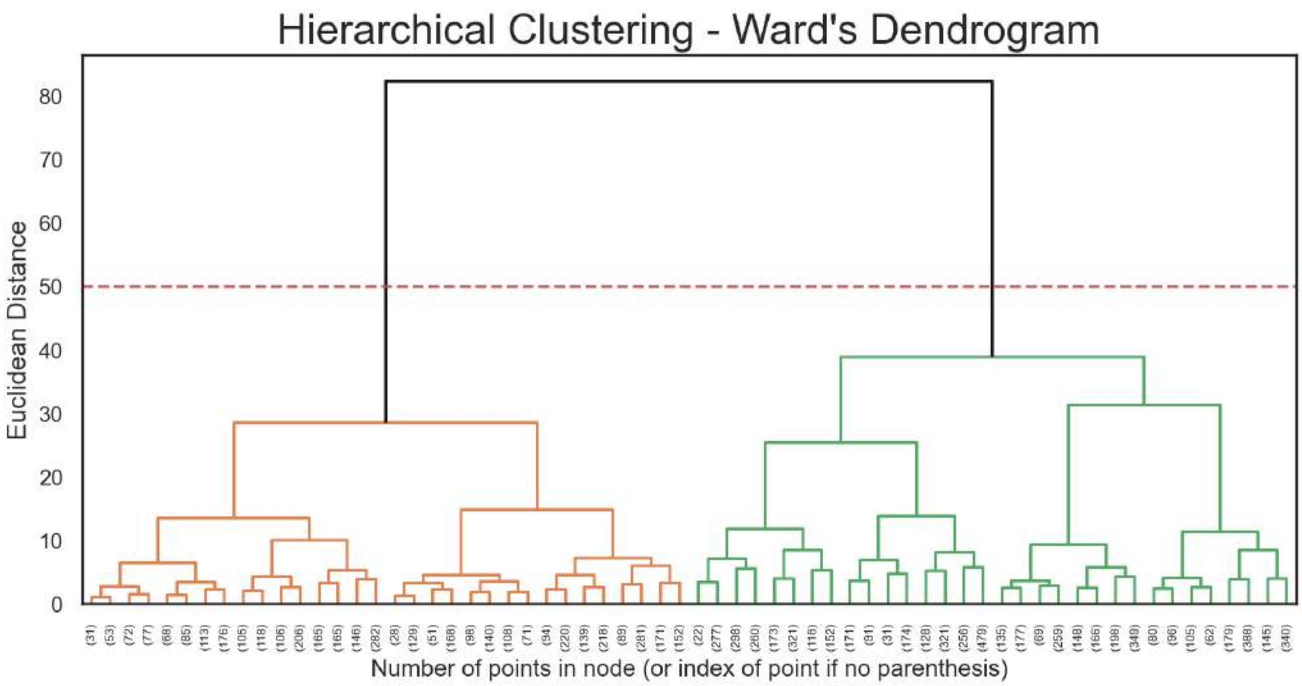
CustID	Age	ClaimsRate	BoxCox_Salary	BoxCox_Salary%SalaryInsurance
1	-0.46875	-0.518519	-0.120988	0.676132
2	-0.87500	0.382716	-0.812801	0.371473
3	-0.09375	-0.654321	-0.078549	0.808431
4	-0.43750	0.222222	-0.603995	0.309289
5	-0.18750	0.111111	-0.300340	0.565863
...	...	...	...	...
10292	0.56250	0.185185	0.295579	0.756852
10293	0.46875	-1.000000	-0.013785	0.821071
10294	-0.28125	-0.740741	0.186797	0.729826
10295	-0.31250	-0.197531	-0.209452	0.767775
10296	-0.43750	-0.666667	0.144867	0.710838

10293 rows × 4 columns

In [271]:
```
linkage = 'ward'
distance = 'euclidean'
hclust = AgglomerativeClustering(linkage=linkage, affinity=distance, distance_threshold=0, n_clusters=None)
hclust.fit_predict(demographic_kmeans)
```

Out[271]:
```
array([8493, 6955, 5915, ...,    3,    1,    0], dtype=int64)
```

In [272]:
```
# Plotting dendrogram
counts = np.zeros(hclust.children_.shape[0])
n_samples = len(hclust.labels_)


for i, merge in enumerate(hclust.children_):
    current_count = 0
    for child_idx in merge:
        if child_idx < n_samples:
            current_count += 1
        else:
            current_count += counts[child_idx - n_samples]
    counts[i] = current_count

linkage_matrix = np.column_stack(
    [hclust.children_, hclust.distances_, counts]
).astype(float)


fig = plt.figure(figsize=(11,5))

y_threshold = 50
dendrogram(linkage_matrix, truncate_mode='level', p=5, color_threshold=y_threshold, above_threshold_color='k')
plt.hlines(y_threshold, 0, 1000, colors="r", linestyles="dashed")
plt.title(f'Hierarchical Clustering - {linkage.title()}\'s Dendrogram', fontsize=21)
plt.xlabel('Number of points in node (or index of point if no parenthesis)')
plt.ylabel(f'{distance.title()} Distance', fontsize=13)

plt.show()
```



In [273]:
```
linkage = 'ward'
distance = 'euclidean'
num_clust = 2

final_hclust = AgglomerativeClustering(affinity = distance, linkage = linkage, n_clusters = num_clust)
final_hc_labels = final_hclust.fit_predict(demographic_kmeans)

demographic_kmeans['label'] = final_hc_labels
demographic_kmeans.groupby('label').mean()
```

Out[273]:

label	Age	ClaimsRate	BoxCox_Salary	BoxCox_Salary%SalaryInsurance
0	-0.397034	-0.172811	-0.270078	0.557603
1	0.519523	-0.142527	0.403641	0.828963

In [274]:
```
print(len(final_hc_labels))
print(len(demographic_kmeans))
```

```
10293
10293
```

In [275]:
```
demographic_kmeans
```

Out[275]:

CustID	Age	ClaimsRate	BoxCox_Salary	BoxCox_Salary%SalaryInsurance	label
1	-0.46875	-0.518519	-0.120988	0.676132	0
2	-0.87500	0.382716	-0.812801	0.371473	0
3	-0.09375	-0.654321	-0.078549	0.808431	0
4	-0.43750	0.222222	-0.603995	0.309289	0
5	-0.18750	0.111111	-0.300340	0.565863	0
...	...	...	...	...	...
10292	0.56250	0.185185	0.295579	0.756852	1
10293	0.46875	-1.000000	-0.013785	0.821071	1
10294	-0.28125	-0.740741	0.186797	0.729826	0
10295	-0.31250	-0.197531	-0.209452	0.767775	0
10296	-0.43750	-0.666667	0.144867	0.710838	0

10293 rows × 5 columns

In [276]:
```
demographic_kmeans.drop('label', axis=1, inplace=True)
```

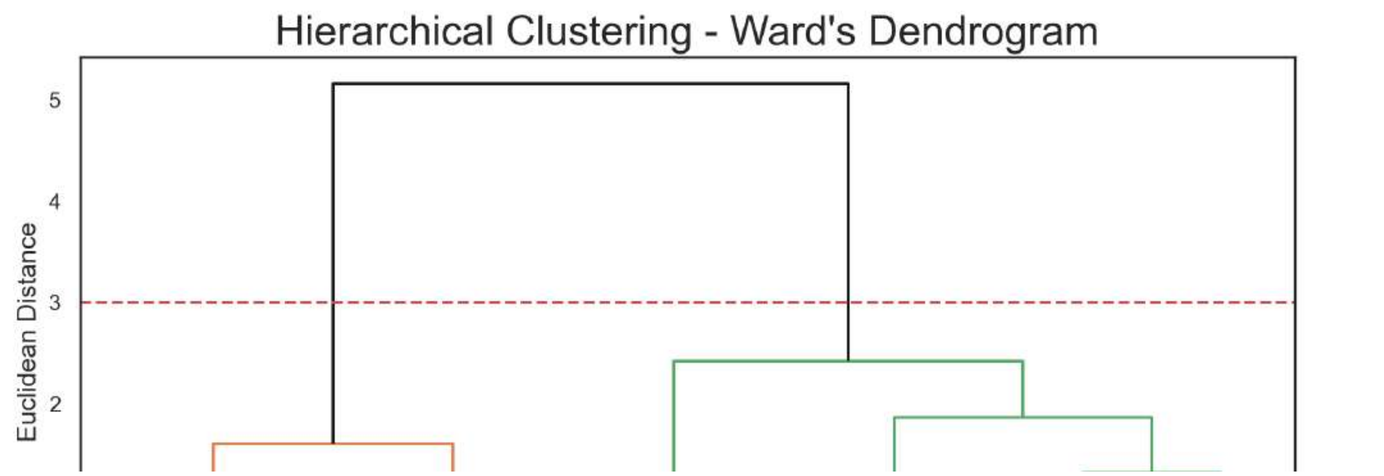
In [277]:
```
def get_r2_scores(df, clusterer, min_k=1, max_k=8):
    """
    Loop over different values of k. To be used with sklearn clusterers.
    """
    r2_clust = {}
    for n in range(min_k, max_k):
        clust = clone(clusterer).set_params(n_clusters=n)
        labels = clust.fit_predict(df)
        r2_clust[n] = r2(df, labels)
    return r2_clust
```

In [278]:
```
#get_r2_scores(demographic_kmeans, hierarchical.set_params(linkage=linkage), 2,6)
```

In [279]:
```
linkage = 'ward'
distance = 'euclidean'
hclust = AgglomerativeClustering(linkage=linkage, affinity=distance, distance_threshold=0, n_clusters=None)
hclust.fit_predict(meanmixed)
```

Out[279]:
```
array([22, 33, 21, 31, 24, 29, 27, 17, 28, 26, 30, 19, 20, 32, 25, 18, 23,
       34, 11,  8, 16,  9, 12,  5, 15,  7, 14, 13, 10,  4,  6,  3,  1,  2,
        0], dtype=int64)
```

In [280]:
```python
# Plotting dendrogram
counts = np.zeros(hclust.children_.shape[0])
n_samples = len(hclust.labels_)


for i, merge in enumerate(hclust.children_):
    current_count = 0
    for child_idx in merge:
        if child_idx < n_samples:
            current_count += 1
        else:
            current_count += counts[child_idx - n_samples]
    counts[i] = current_count

linkage_matrix = np.column_stack(
    [hclust.children_, hclust.distances_, counts]
).astype(float)


fig = plt.figure(figsize=(11,5))

y_threshold = 3
dendrogram(linkage_matrix, truncate_mode='level', p=5, color_threshold=y_threshold, above_threshold_color='k')
plt.hlines(y_threshold, 0, 1000, colors="r", linestyles="dashed")
plt.title(f'Hierarchical Clustering - {linkage.title()}\'s Dendrogram', fontsize=21)
plt.xlabel('Number of points in node (or index of point if no parenthesis)')
plt.ylabel(f'{distance.title()} Distance', fontsize=13)

plt.show()
```



In [281]:
```python
linkage = 'ward'
distance = 'euclidean'
num_clust = 2

final_hclust = AgglomerativeClustering(affinity = distance, linkage = linkage, n_clusters = num_clust)
final_hc_labels = final_hclust.fit_predict(meanmixed)

meanmixed['label'] = final_hc_labels
meanmixed.groupby(meanmixed['label']).mean()
```

Out[281]:

label	Age	ClaimsRate	BoxCox_Salary	BoxCox_Salary%SalaryInsurance
0	-0.488673	-0.246412	-0.367194	0.481371
1	0.467531	-0.080455	0.364436	0.812030

In [282]:
```python
print(len(final_hc_labels))
print(len(meanmixed))
```

```
35
35
```

In [283]:
```
meanmixed
```

Out[283]:

label	Age	ClaimsRate	BoxCox_Salary	BoxCox_Salary%SalaryInsurance	label
0	-0.164569	-0.569460	-0.000423	0.677952	0
1	0.674260	0.276348	0.474091	0.850652	1
2	-0.774359	-0.163484	-0.552283	0.528701	0
3	-0.115132	0.243165	-0.247396	0.505477	0
4	0.166221	-0.007974	0.104513	0.767506	1
5	0.801941	-0.257110	0.656988	0.889027	1
6	0.577427	-0.365715	0.434947	0.858920	1
7	-0.814312	-0.641260	-0.833854	-0.112467	0
8	-0.585653	-0.132304	-0.243976	0.672245	0
9	-0.349130	0.216282	-0.137117	0.639704	0
10	0.159824	-0.755403	0.217323	0.734051	0
11	0.666972	-0.041671	0.473596	0.863740	1
12	-0.850922	0.323872	-0.522317	0.617412	0
13	-0.797945	0.162185	-0.897091	-0.293265	0
14	-0.465909	-0.579529	-0.173802	0.648241	0
15	-0.350953	-0.268025	-0.365261	0.558415	0
16	0.389596	-0.149735	0.292286	0.834164	1
17	-0.744591	-0.208452	-0.711563	0.210704	0
18	0.456871	-0.664717	0.366301	0.806863	1
19	-0.444433	0.235163	-0.433455	0.420510	0
20	0.481696	0.210053	0.223618	0.770849	1
21	-0.647211	0.167024	-0.321421	0.634316	0
22	-0.846721	-0.671455	-0.618912	0.497413	0
23	-0.810409	0.224471	-0.697086	0.309320	0
24	0.771844	-0.610595	0.583832	0.861370	1
25	0.161643	0.269795	0.172737	0.716821	1
26	0.816935	0.195141	0.676448	0.877548	1
27	-0.194473	-0.137639	-0.057362	0.722454	0
28	0.023821	-0.802118	0.000066	0.598501	0
29	-0.755743	-0.497309	-0.399227	0.650161	0
30	-0.046524	0.246257	0.022338	0.641460	1
31	0.258690	-0.436364	0.181061	0.786636	1
32	0.367863	0.209915	0.439345	0.842868	1
33	-0.220433	-0.726353	-0.259103	0.493749	0
34	-0.512876	-0.594023	-0.456807	0.395205	0

In [284]:
```
meanmixed.drop('label',axis=1,inplace=True)
```

In [285]:
```
def get_r2_scores(df, clusterer, min_k=1, max_k=8):
    """
    Loop over different values of k. To be used with sklearn clusterers.
    """
    r2_clust = {}
    for n in range(min_k, max_k):
        clust = clone(clusterer).set_params(n_clusters=n)
        labels = clust.fit_predict(df)
        r2_clust[n] = r2(df, labels)
    return r2_clust
```

In [286]:
```
#get_r2_scores(meanmixed, hierarchical.set_params(linkage=linkage), 2,6)
```

Merging the perspectives

[Back to Index](#)

In [287]:
```
mergedAz = value.join(demographic_kmeans)
mergedAz
```

Out[287]:

CustID	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal	Age	ClaimsRate	Bo
1	-0.009429	-0.021396	0.349075	0.244747	-0.019899	0.584111	-0.46875	-0.518519	
2	0.471464	-0.173807	-0.663698	0.818614	0.539493	0.018863	-0.87500	0.382716	
3	0.252744	-0.132395	-0.207693	0.470097	0.512140	0.680768	-0.09375	-0.654321	
4	-0.105429	0.717236	-0.288574	0.150285	0.101420	0.179923	-0.43750	0.222222	
5	-0.092433	0.154300	0.229733	-0.031379	0.209041	0.244560	-0.18750	0.111111	
...	...	...	...	...	...	...	...	...	
10292	-0.087709	0.112621	0.405970	-0.176866	-0.046505	0.201349	0.56250	0.185185	
10293	0.878771	-0.036971	-0.459338	-0.120570	0.535732	0.406156	0.46875	-1.000000	
10294	0.103840	-0.041997	0.437304	-0.124305	-0.200621	0.695109	-0.28125	-0.740741	
10295	0.233593	0.227932	-0.267597	0.354532	0.564307	0.469815	-0.31250	-0.197531	
10296	0.025238	-0.047033	0.470325	-0.237410	-0.071846	0.649855	-0.43750	-0.666667	

10293 rows × 10 columns

In [288]:
```
# Applying the right clustering (algorithm and number of clusters) for each perspective

kmeans_demogra = KMeans(n_clusters=3, init='k-means++', n_init=15, random_state=1)

# demo_lab = kmeans_demogra.fit_predict(demographic, categorical=categorical_columns)
demo_lab = kmeans_demogra.fit_predict(demographic_kmeans)

kmeans_value = KMeans(
    n_clusters=4,
    init='k-means++',
    n_init=20,
    random_state=0
)
value_lab = kmeans_value.fit_predict(value)

mergedAz['demographic'] = demo_lab
mergedAz['value'] = value_lab
```

In [289]:
```
AzMetricF = ['BoxCox_PremHousehold', 'BoxCox_PremHealth', 'BoxCox_PremMotor', 'BoxCox_PremLife', 'BoxCox_PremWork', 'BoxCox_CustM
```

In [290]:
```
mergedAz
```

Out[290]:

CustID	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal	Age	ClaimsRate
1	-0.009429	-0.021396	0.349075	0.244747	-0.019899	0.584111	-0.46875	-0.518519
2	0.471464	-0.173807	-0.663698	0.818614	0.539493	0.018863	-0.87500	0.382716
3	0.252744	-0.132395	-0.207693	0.470097	0.512140	0.680768	-0.09375	-0.654321
4	-0.105429	0.717236	-0.288574	0.150285	0.101420	0.179923	-0.43750	0.222222
5	-0.092433	0.154300	0.229733	-0.031379	0.209041	0.244560	-0.18750	0.111111
...	...	...	...	...	...	...	...	...
10292	-0.087709	0.112621	0.405970	-0.176866	-0.046505	0.201349	0.56250	0.185185
10293	0.878771	-0.036971	-0.459338	-0.120570	0.535732	0.406156	0.46875	-1.000000
10294	0.103840	-0.041997	0.437304	-0.124305	-0.200621	0.695109	-0.28125	-0.740741
10295	0.233593	0.227932	-0.267597	0.354532	0.564307	0.469815	-0.31250	-0.197531

In [291]:
```
# Count label frequencies (contigency table)
mergedAz.groupby(['value', 'demographic'])\
    .size()\
    .to_frame()\
    .reset_index()\
    .pivot('demographic', 'value', 0)
```

Out[291]:

value / demographic	0	1	2	3
0	795	180	135	989
1	1178	1261	1132	592
2	1478	506	1235	812

In [292]:
```
# Clusters with low frequency to be merged:
to_merge = [(0,1), (0,2), (2,1),(1,3)]

df_centroids = mergedAz.groupby(['demographic', 'value'])\
    [AzMetricF].mean()

# Computing the euclidean distance matrix between the centroids
euclidean = pairwise_distances(df_centroids)
df_dists = pd.DataFrame(
    euclidean, columns=df_centroids.index, index=df_centroids.index
)

# Merging each low frequency clustering (source) to the closest cluster (target)
source_target = {}
for clus in to_merge:
    if clus not in source_target.values():
        source_target[clus] = df_dists.loc[clus].sort_values().index[1]

source_target
```

Out[292]:
```
{(0, 1): (0, 2), (2, 1): (2, 2), (1, 3): (2, 3)}
```

In [293]:
```
mergedAz_ = mergedAz.copy()

# Changing the behavior_labels and product_labels based on source_target
for source, target in source_target.items():
    mask = (mergedAz_['demographic']==source[0]) & (mergedAz_['value']==source[1])
    mergedAz_.loc[mask, 'demographic'] = target[0]
    mergedAz_.loc[mask, 'value'] = target[1]

# New contigency table
mergedAz_.groupby(['value', 'demographic'])\
    .size()\
    .to_frame()\
    .reset_index()\
    .pivot('demographic', 'value', 0)
```

Out[293]:

value	0	1	2	3
demographic				
0	795.0	NaN	315.0	989.0
1	1178.0	1261.0	1132.0	NaN
2	1478.0	NaN	1741.0	1404.0

In [294]:
```
# Centroids of the concatenated cluster labels
df_centroids = mergedAz_.groupby(['demographic', 'value'])\
    [AzMetricF].mean()
df_centroids
```

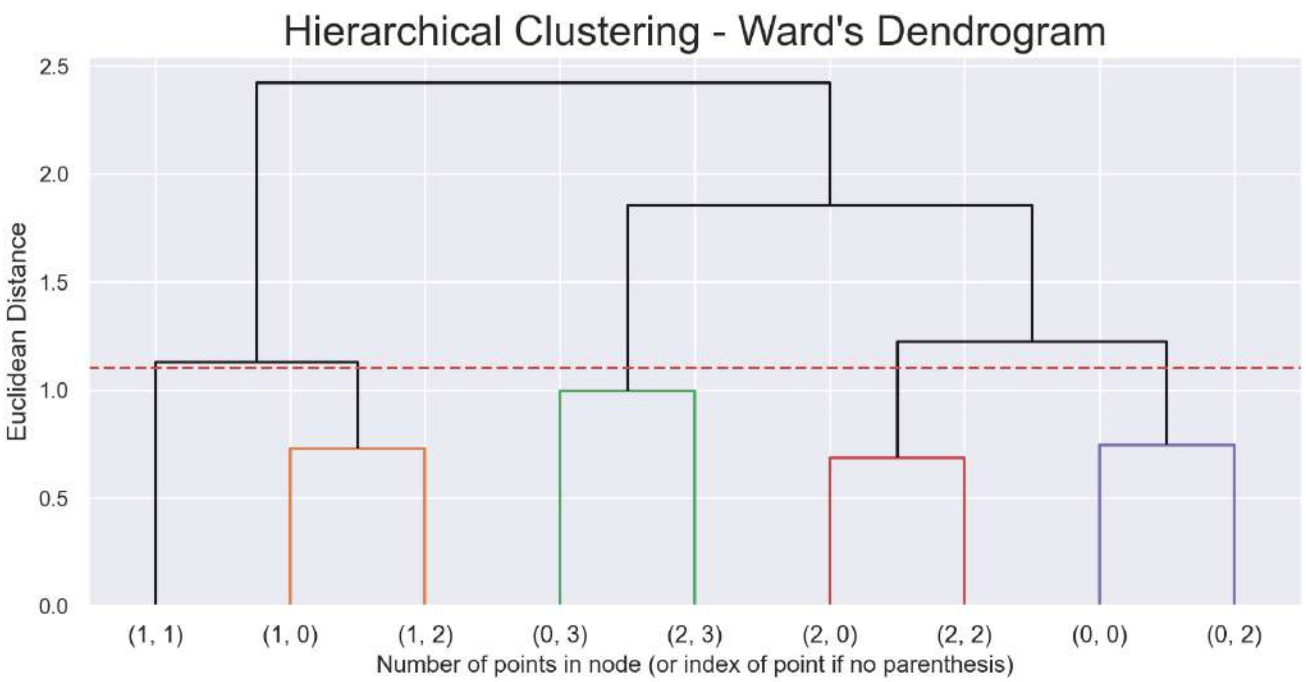
Out[294]:

		BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal	BoxCox_Sal
demographic	**value**							
	0	0.074437	-0.015840	0.268389	0.003493	0.037290	0.610766	0.0689
0	**2**	0.421708	0.123579	-0.250642	0.274833	0.276688	0.681549	0.0620
	3	-0.183312	-0.380948	0.682585	-0.283282	-0.198213	0.648571	0.0426
	0	0.052368	0.093979	0.216924	-0.014137	0.021571	0.353288	-0.3070
1	**1**	0.505511	-0.010849	-0.576511	0.492984	0.466722	0.415430	-0.5542
	2	0.186838	0.507134	-0.284541	0.154612	0.187918	0.355714	-0.4140
	0	0.077298	0.051823	0.227339	0.017272	0.056573	0.342127	0.3821
2	**2**	0.252460	0.359305	-0.267185	0.245879	0.224889	0.366838	0.4459
	3	-0.171090	-0.332165	0.652919	-0.281036	-0.195478	0.217237	0.0503

In [295]:
```
# Using Hierarchical clustering to merge the concatenated cluster centroids
hclust = AgglomerativeClustering(
    linkage='ward',
    affinity='euclidean',
    distance_threshold=0,
    n_clusters=None
)
hclust_labels = hclust.fit_predict(df_centroids)
```

In [296]:

```
# Adapted from:
# https://scikit-learn.org/stable/auto_examples/cluster/plot_agglomerative_dendrogram.html#sphx-glr-auto-examples-cluster-plot-ag

# create the counts of samples under each node (number of points being merged)
counts = np.zeros(hclust.children_.shape[0])
n_samples = len(hclust.labels_)

# hclust.children_ contains the observation ids that are being merged together
# At the i-th iteration, children[i][0] and children[i][1] are merged to form node n_samples + i
for i, merge in enumerate(hclust.children_):
    # track the number of observations in the current cluster being formed
    current_count = 0
    for child_idx in merge:
        if child_idx < n_samples:
            # If this is True, then we are merging an observation
            current_count += 1  # leaf node
        else:
            # Otherwise, we are merging a previously formed cluster
            current_count += counts[child_idx - n_samples]
    counts[i] = current_count

# the hclust.children_ is used to indicate the two points/clusters being merged (dendrogram's u-joins)
# the hclust.distances_ indicates the distance between the two points/clusters (height of the u-joins)
# the counts indicate the number of points being merged (dendrogram's x-axis)
linkage_matrix = np.column_stack(
    [hclust.children_, hclust.distances_, counts]
).astype(float)

# Plot the corresponding dendrogram
sns.set()
fig = plt.figure(figsize=(11,5))
# The Dendrogram parameters need to be tuned
y_threshold = 1.1
dendrogram(linkage_matrix, truncate_mode='level', labels=df_centroids.index, p=5, color_threshold=y_threshold, above_threshold_co
plt.hlines(y_threshold, 0, 1000, colors="r", linestyles="dashed")
plt.title(f'Hierarchical Clustering - {linkage.title()}\'s Dendrogram', fontsize=21)
plt.xlabel('Number of points in node (or index of point if no parenthesis)')
plt.ylabel(f'Euclidean Distance', fontsize=13)
plt.show()
```



In [297]:

```
cluster_changes = {
 (0, 0): 3,
 (0, 1): 3,
 (0, 2): 3,
 (0, 3): 4,
 (1, 0): 7,
 (1, 1): 6,
 (1, 2): 7,
 (1, 3): 12,
 (2, 0): 11,
 (2, 1): 11,
 (2, 2): 11,
 (2, 3): 12}
```

In [298]:
```python
mergedAz_['merged_labels'] = mergedAz_.apply(
    lambda row: cluster_changes[
        (row['demographic'], row['value'])
    ], axis=1
)

mergedAz_.groupby('merged_labels').mean()
```

Out[298]:

merged_labels	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal	Age	Claims
3	0.172987	0.023725	0.121097	0.080495	0.105227	0.630853	0.046368	-0.56
4	-0.183312	-0.380948	0.682585	-0.283282	-0.198213	0.648571	0.006920	-0.75
6	0.505511	-0.010849	-0.576511	0.492984	0.466722	0.415430	-0.760061	-0.13
7	0.118264	0.296443	-0.028816	0.068557	0.103088	0.354477	-0.523390	-0.06
11	0.172034	0.218125	-0.040125	0.140914	0.147607	0.355492	0.544589	-0.05
12	-0.171090	-0.332165	0.652919	-0.281036	-0.195478	0.217237	0.035791	0.14

In [299]:
```python
mergedAz_['merged_labels'].value_counts()
```

Out[299]:
```
11    3219
7     2310
12    1404
6     1261
3     1110
4      989
Name: merged_labels, dtype: int64
```

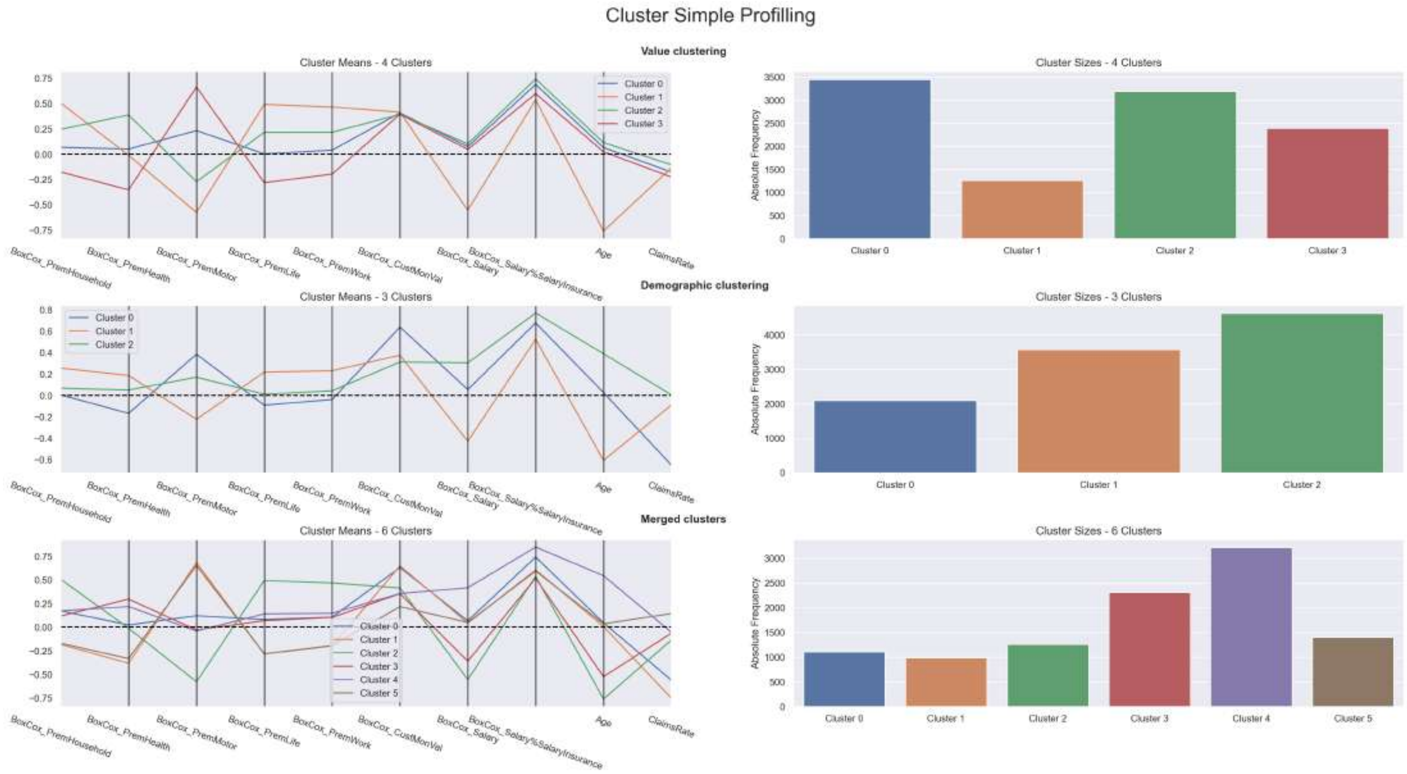
Cluster Analysis

Back to Index

Parallel Coordinate and Frequency (Value/Deemographic/Merged)

In [300]:
```python
def cluster_profiles(df, label_columns, figsize, compar_titles=None):
    """
    Pass df with labels columns of one or multiple clustering labels.
    Then specify this label columns to perform the cluster profile according to them.
    """
    if compar_titles == None:
        compar_titles = [""]*len(label_columns)

    sns.set()
    fig, axes = plt.subplots(nrows=len(label_columns), ncols=2, figsize=figsize, squeeze=False)
    for ax, label, titl in zip(axes, label_columns, compar_titles):
        # Filtering df
        drop_cols = [i for i in label_columns if i!=label]
        dfax = df.drop(drop_cols, axis=1)

        # Getting the cluster centroids and counts
        centroids = dfax.groupby(by=label, as_index=False).mean()
        counts = dfax.groupby(by=label, as_index=False).count().iloc[:,[0,1]]
        counts.columns = [label, "counts"]

        # Setting Data
        pd.plotting.parallel_coordinates(centroids, label, color=sns.color_palette(), ax=ax[0])
        sns.barplot(x=label, y="counts", data=counts, ax=ax[1])

        #Setting Layout
        handles, _ = ax[0].get_legend_handles_labels()
        cluster_labels = ["Cluster {}".format(i) for i in range(len(handles))]
        ax[0].annotate(text=titl, xy=(0.95,1.1), xycoords='axes fraction', fontsize=13, fontweight = 'heavy')
        ax[0].legend(handles, cluster_labels) # Adaptable to number of clusters
        ax[0].axhline(color="black", linestyle="--")
        ax[0].set_title("Cluster Means - {} Clusters".format(len(handles)), fontsize=13)
        ax[0].set_xticklabels(ax[0].get_xticklabels(), rotation=-20)
        ax[1].set_xticklabels(cluster_labels)
        ax[1].set_xlabel("")
        ax[1].set_ylabel("Absolute Frequency")
        ax[1].set_title("Cluster Sizes - {} Clusters".format(len(handles)), fontsize=13)

    plt.subplots_adjust(hspace=0.4, top=0.90)
    plt.suptitle("Cluster Simple Profilling", fontsize=23)
    plt.show()
```

In [301]:
```python
# Profilling each cluster (product, behavior, merged)
cluster_profiles(
    df = mergedAz_[AzMetricF + ['value', 'demographic', 'merged_labels']],
    label_columns = ['value', 'demographic', 'merged_labels'],
    figsize = (28, 13),
    compar_titles = ["Value clustering", "Demographic clustering", "Merged clusters"]
)
```



In [302]:
```python
mergedAz_
```

Out[302]:

CustID	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal	Age	ClaimsRate	Bo
1	-0.009429	-0.021396	0.349075	0.244747	-0.019899	0.584111	-0.46875	-0.518519	
2	0.471464	-0.173807	-0.663698	0.818614	0.539493	0.018863	-0.87500	0.382716	
3	0.252744	-0.132395	-0.207693	0.470097	0.512140	0.680768	-0.09375	-0.654321	
4	-0.105429	0.717236	-0.288574	0.150285	0.101420	0.179923	-0.43750	0.222222	
5	-0.092433	0.154300	0.229733	-0.031379	0.209041	0.244560	-0.18750	0.111111	
...	...	...	...	...	...	...	...	...	
10292	-0.087709	0.112621	0.405970	-0.176866	-0.046505	0.201349	0.56250	0.185185	
10293	0.878771	-0.036971	-0.459338	-0.120570	0.535732	0.406156	0.46875	-1.000000	
10294	0.103840	-0.041997	0.437304	-0.124305	-0.200621	0.695109	-0.28125	-0.740741	
10295	0.233593	0.227932	-0.267597	0.354532	0.564307	0.469815	-0.31250	-0.197531	
10296	0.025238	-0.047033	0.470325	-0.237410	-0.071846	0.649855	-0.43750	-0.666667	

10293 rows × 13 columns

PCA

In [303]:
```python
Allf = ['BoxCox_PremHousehold', 'BoxCox_PremHealth', 'BoxCox_PremMotor', 'BoxCox_PremLife', 'BoxCox_PremWork', 'BoxCox_CustMonVal
```

In [304]:

```
from sklearn.decomposition import PCA
# Use PCA to reduce dimensionality of data
pca = PCA()
pca_feat = pca.fit_transform(mergedAz_[AzMetricF])
pca_feat  # What is this output?

# Output PCA table
pd.DataFrame(
    {"Eigenvalue": pca.explained_variance_,
     "Difference": np.insert(np.diff(pca.explained_variance_), 0, 0),
     "Proportion": pca.explained_variance_ratio_,
     "Cumulative": np.cumsum(pca.explained_variance_ratio_)},
    index=range(1, pca.n_components_ + 1)
)

# Output PCA table
pd.DataFrame(
    {"Eigenvalue": pca.explained_variance_,
     "Difference": np.insert(np.diff(pca.explained_variance_), 0, 0),
     "Proportion": pca.explained_variance_ratio_,
     "Cumulative": np.cumsum(pca.explained_variance_ratio_)},
    index=range(1, pca.n_components_ + 1)
)
```

Out[304]:

	Eigenvalue	Difference	Proportion	Cumulative
1	0.540836	0.000000	0.387948	0.387948
2	0.364382	-0.176453	0.261376	0.649323
3	0.205698	-0.158684	0.147550	0.796873
4	0.105465	-0.100233	0.075652	0.872524
5	0.063657	-0.041808	0.045662	0.918186
6	0.057241	-0.006417	0.041059	0.959246
7	0.027924	-0.029316	0.020030	0.979276
8	0.020389	-0.007535	0.014626	0.993902
9	0.004930	-0.015459	0.003536	0.997438
10	0.003571	-0.001359	0.002562	1.000000

In [305]:

```
from sklearn.decomposition import PCA
# Perform PCA again with the number of principal components you want to retain
pca = PCA(n_components=4)
pca_feat = pca.fit_transform(mergedAz_[AzMetricF])
pca_feat_names = [f"PC{i}" for i in range(pca.n_components_)]
pca_df = pd.DataFrame(pca_feat, index=mergedAz_.index, columns=pca_feat_names)  # remember index=df_pca.index
pca_df
```

Out[305]:

	PC0	PC1	PC2	PC3
CustID				
1	0.173224	-0.458348	-0.336458	0.002310
2	1.630829	0.084705	0.530198	0.617173
3	0.363409	0.248259	-0.654276	0.356897
4	0.829147	-0.018018	0.549962	-0.612322
5	0.165053	-0.251369	0.381829	-0.117690
...	...	...	...	...
10292	-0.776859	0.133704	0.431760	-0.091101
10293	0.091642	0.633123	-0.926192	0.196049
10294	-0.253892	-0.472511	-0.585510	-0.144182
10295	0.639087	0.291689	-0.120606	0.072326
10296	-0.165272	-0.592487	-0.476962	-0.166739

10293 rows × 4 columns

In [306]:
```
# Reassigning df to contain pca variables
df_pca = pd.concat([mergedAz_, pca_df], axis=1)
df_pca.head()
```

Out[306]:

CustID	BoxCox_PremHousehold	BoxCox_PremHealth	BoxCox_PremMotor	BoxCox_PremLife	BoxCox_PremWork	BoxCox_CustMonVal	Age	ClaimsRate	Bo
1	-0.009429	-0.021396	0.349075	0.244747	-0.019899	0.584111	-0.46875	-0.518519	
2	0.471464	-0.173807	-0.663698	0.818614	0.539493	0.018863	-0.87500	0.382716	
3	0.252744	-0.132395	-0.207693	0.470097	0.512140	0.680768	-0.09375	-0.654321	
4	-0.105429	0.717236	-0.288574	0.150285	0.101420	0.179923	-0.43750	0.222222	
5	-0.092433	0.154300	0.229733	-0.031379	0.209041	0.244560	-0.18750	0.111111	

In [307]:
```
PC = ['PC0', 'PC1', 'PC2', 'PC3']
```

In [308]:
```
colour = mergedAz_['merged_labels']
```

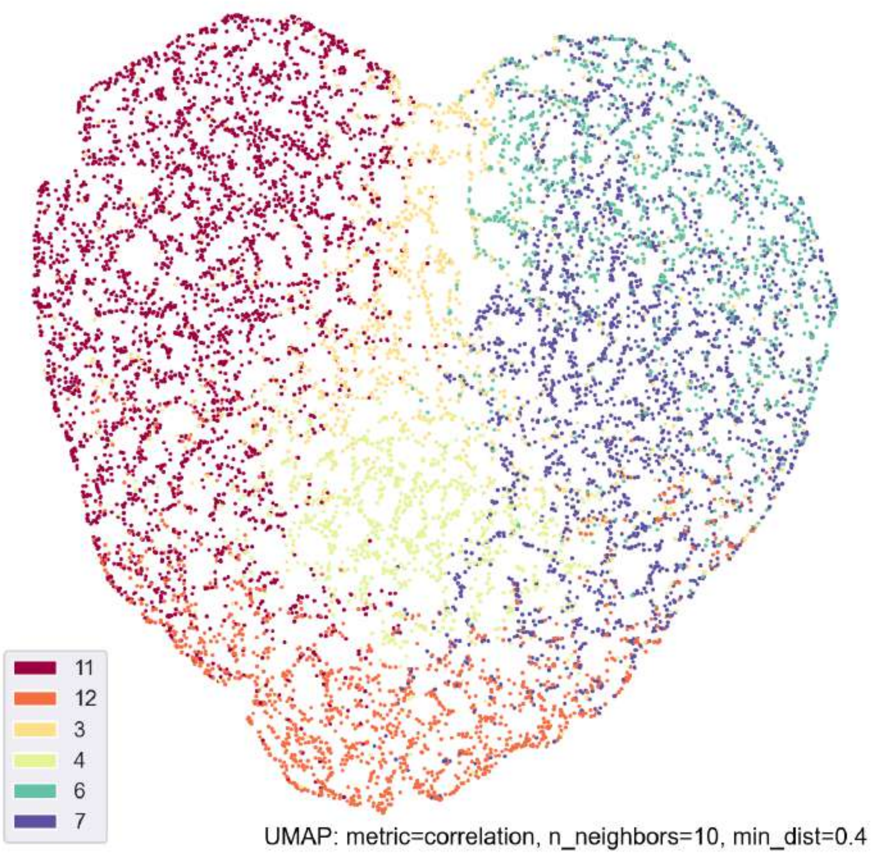
UMAP

In [311]:
```
hue = mergedAz_['merged_labels'].astype('str')
```

In [312]:
```
embedding = umap.UMAP(n_neighbors=10,
                      min_dist=0.4,
                      metric='correlation').fit(df_pca[PC])
```

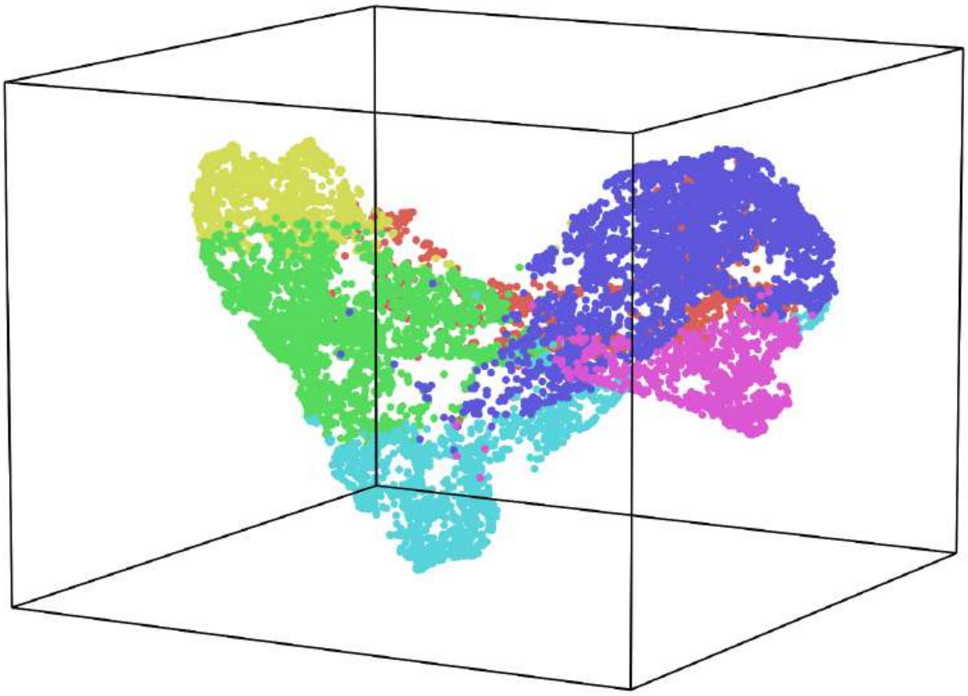
In [313]:
```
umap.plot.points(embedding, labels=hue)
```

Out[313]: <AxesSubplot:>



In [314]:
```
#methods = ['PCA', 'IncrementalPCA', 'SparsePCA', 'MiniBatchSparsePCA', 'KernelPCA', 'FastICA', 'FactorAnalysis', 'TruncatedSVD',
methods = ['UMAP']
hue = mergedAz_['merged_labels'].astype('str')

for param in methods:
    hyp.plot(df_pca[PC], '.', reduce=param, hue=hue, ndims=3)


    print(param)
```

UMAP



T-SNE

In [315]:
```
two_dim = TSNE(random_state=42,perplexity = 100).fit_transform(df_pca[PC])

pd.DataFrame(two_dim).plot.scatter(x=0, y=1, c=colour, colormap='tab10', figsize=(15,10))
plt.title("TSNE", fontsize=14)
plt.show()
```