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

filename = 'Prudential_Life_Insurance_train.csv'
```

Data Acquisition

In [ ]:
```
df = pd.read_csv(filename)
display(df.head(10))
```

	Id	Product_Info_1	Product_Info_2	Product_Info_3	Product_Info_4	Product_Info_5	Pr
0	2	1	D3	10	0.076923	2	
1	5	1	A1	26	0.076923	2	
2	6	1	E1	26	0.076923	2	
3	7	1	D4	10	0.487179	2	
4	8	1	D2	26	0.230769	2	
5	10	1	D2	26	0.230769	3	
6	11	1	A8	10	0.166194	2	
7	14	1	D2	26	0.076923	2	
8	15	1	D3	26	0.230769	2	
9	16	1	E1	21	0.076923	2	

10 rows × 128 columns

In [ ]:
```
display(df.info())
display(df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 128 entries, Id to Response
dtypes: float64(18), int64(109), object(1)
memory usage: 58.0+ MB
None
```

	Id	Product_Info_1	Product_Info_3	Product_Info_4	Product_Info_5	Pro
count	59381.000000	59381.000000	59381.000000	59381.000000	59381.000000	59
mean	39507.211515	1.026355	24.415655	0.328952	2.006955	
std	22815.883089	0.160191	5.072885	0.282562	0.083107	
min	2.000000	1.000000	1.000000	0.000000	2.000000	
25%	19780.000000	1.000000	26.000000	0.076923	2.000000	
50%	39487.000000	1.000000	26.000000	0.230769	2.000000	
75%	59211.000000	1.000000	26.000000	0.487179	2.000000	
max	79146.000000	2.000000	38.000000	1.000000	3.000000	

8 rows × 127 columns

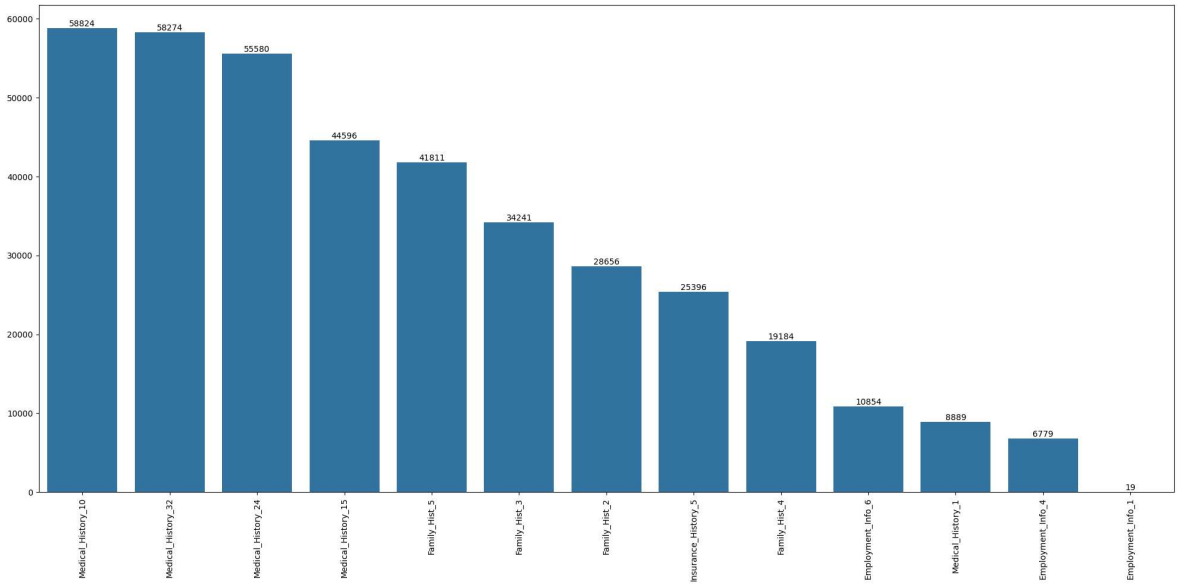
In [ ]:
```
print('There are {} duplicated rows'.format(df.duplicated().sum()))
print('There are {} rows with null values'.format(df.isnull().any(axis=1).sum()))
```

```
There are 0 duplicated rows
There are 59381 rows with null values
```

The dataset has 59381 rows and 128 columns, with float64, int64 and object data types. There is at least a null value in a field for every row, so we will need to investigate further the distribution of this kind of values across the features, through visualization.

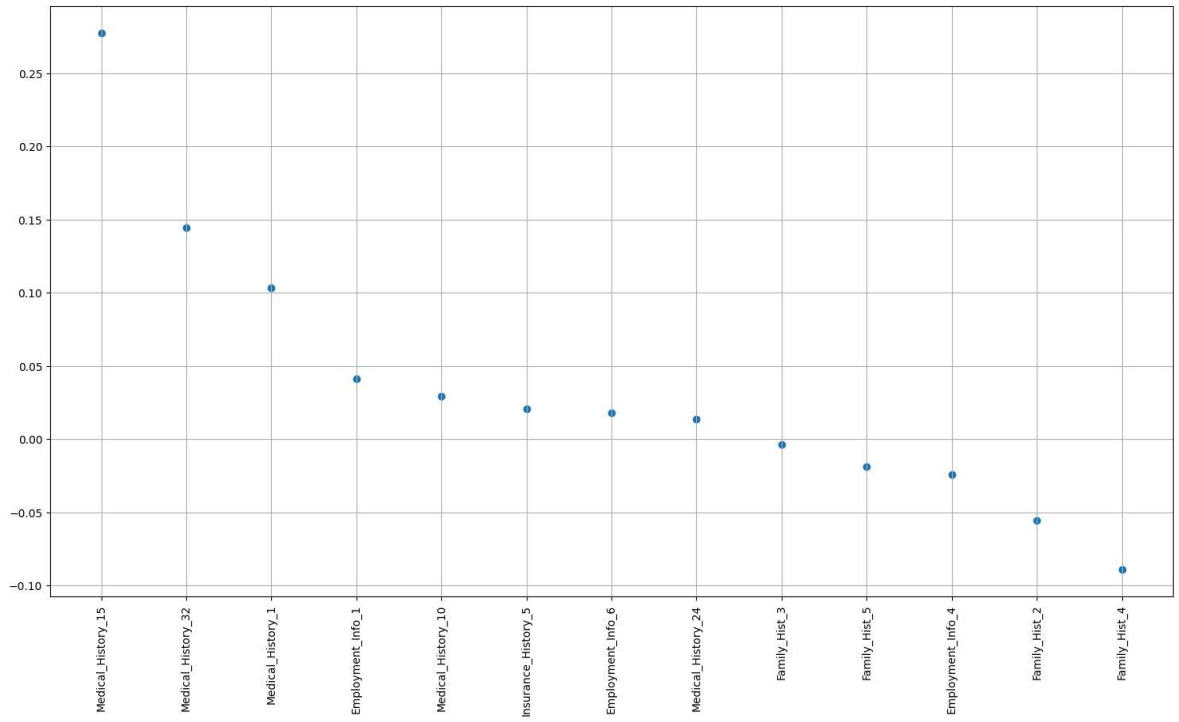
Data Visualization

In [ ]:
```
plt.figure(figsize=(20, 10))

null_c = df.isnull().sum().sort_values(ascending = False)
null_c = null_c[null_c > 0]

sns.barplot(null_c)

for i, v in enumerate(null_c.values):
    plt.text(i, v, str(v), ha='center', va='bottom')

plt.xticks(rotation = 90)
plt.tight_layout()
plt.show()
```



The barplot shows the number of null values across the columns. It can be seen that for the first five columns the number of nulls is greater or equal to the 70% of the total rows in the dataset. To better decide how to treat these values, we will investigate the correlation of these columns with the target.

In [ ]:
```
corr_target = df[null_c.index].corrwith(df['Response']).sort_values(ascending=Fa
plt.figure(figsize=(15, 8))
plt.scatter(corr_target.index,corr_target)
plt.tight_layout()
plt.xticks(rotation = 90)
plt.grid(True)
plt.show()
```



From the chart above it can be seen that none of the features with nulls exhibit great correlation with the target; this suggests that removing some of these columns won't result in a significant loss of information.

In [ ]:
```
nominal_columns = []
quantitative_columns = []

for c in df.columns:
    if df[c].dtype == 'object':
        print('object: ', c)
    if df[c].dtype == 'int64':
        print('int64: ', c)
        nominal_columns.append(c)
    if df[c].dtype == 'float64':
        print('float64: ', c)
        quantitative_columns.append(c)
```

```
int64:  Id
int64:  Product_Info_1
object:  Product_Info_2
int64:  Product_Info_3
float64:  Product_Info_4
int64:  Product_Info_5
int64:  Product_Info_6
int64:  Product_Info_7
float64:  Ins_Age
float64:  Ht
float64:  Wt
float64:  BMI
float64:  Employment_Info_1
int64:  Employment_Info_2
int64:  Employment_Info_3
float64:  Employment_Info_4
int64:  Employment_Info_5
float64:  Employment_Info_6
int64:  InsuredInfo_1
int64:  InsuredInfo_2
int64:  InsuredInfo_3
int64:  InsuredInfo_4
int64:  InsuredInfo_5
int64:  InsuredInfo_6
int64:  InsuredInfo_7
int64:  Insurance_History_1
int64:  Insurance_History_2
int64:  Insurance_History_3
int64:  Insurance_History_4
float64:  Insurance_History_5
int64:  Insurance_History_7
int64:  Insurance_History_8
int64:  Insurance_History_9
int64:  Family_Hist_1
float64:  Family_Hist_2
float64:  Family_Hist_3
float64:  Family_Hist_4
float64:  Family_Hist_5
float64:  Medical_History_1
int64:  Medical_History_2
int64:  Medical_History_3
int64:  Medical_History_4
int64:  Medical_History_5
int64:  Medical_History_6
int64:  Medical_History_7
int64:  Medical_History_8
int64:  Medical_History_9
float64:  Medical_History_10
int64:  Medical_History_11
int64:  Medical_History_12
int64:  Medical_History_13
int64:  Medical_History_14
float64:  Medical_History_15
int64:  Medical_History_16
int64:  Medical_History_17
int64:  Medical_History_18
int64:  Medical_History_19
int64:  Medical_History_20
int64:  Medical_History_21
int64:  Medical_History_22
```

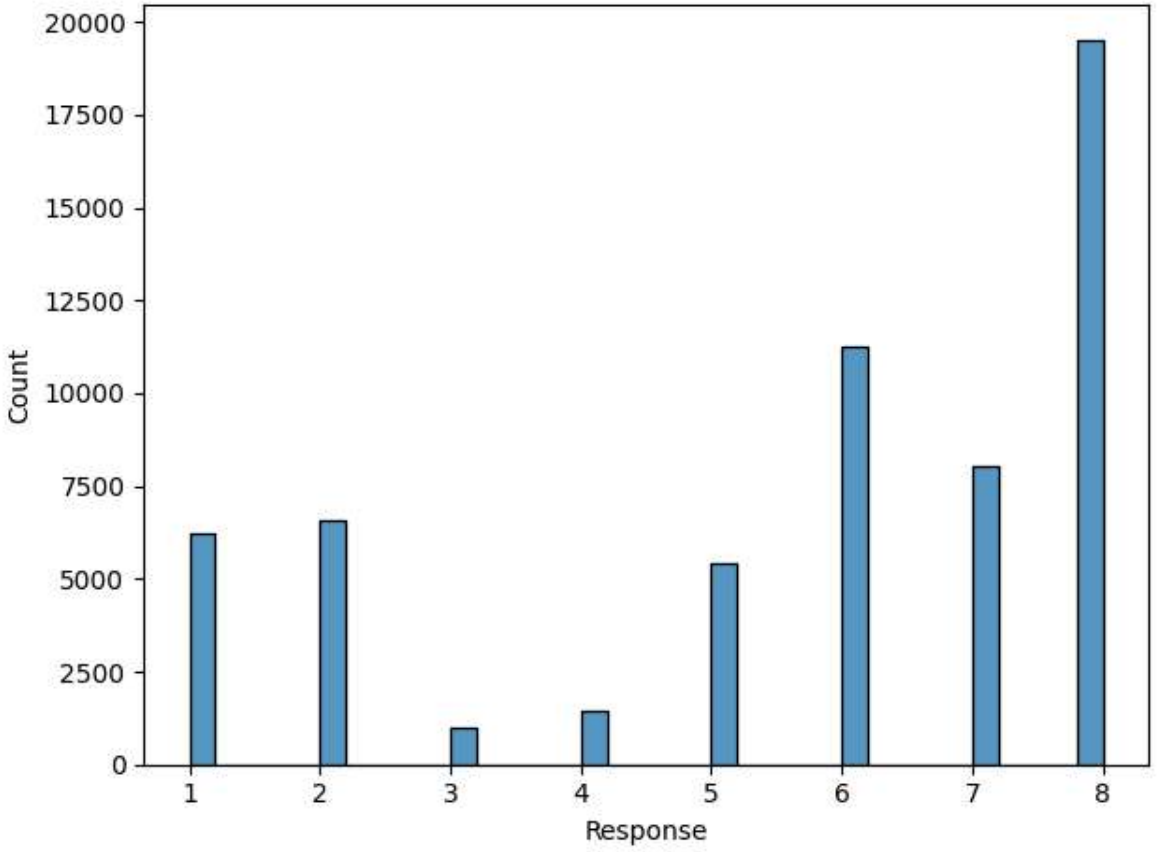
```
int64:  Medical_History_23
float64:  Medical_History_24
int64:  Medical_History_25
int64:  Medical_History_26
int64:  Medical_History_27
int64:  Medical_History_28
int64:  Medical_History_29
int64:  Medical_History_30
int64:  Medical_History_31
float64:  Medical_History_32
int64:  Medical_History_33
int64:  Medical_History_34
int64:  Medical_History_35
int64:  Medical_History_36
int64:  Medical_History_37
int64:  Medical_History_38
int64:  Medical_History_39
int64:  Medical_History_40
int64:  Medical_History_41
int64:  Medical_Keyword_1
int64:  Medical_Keyword_2
int64:  Medical_Keyword_3
int64:  Medical_Keyword_4
int64:  Medical_Keyword_5
int64:  Medical_Keyword_6
int64:  Medical_Keyword_7
int64:  Medical_Keyword_8
int64:  Medical_Keyword_9
int64:  Medical_Keyword_10
int64:  Medical_Keyword_11
int64:  Medical_Keyword_12
int64:  Medical_Keyword_13
int64:  Medical_Keyword_14
int64:  Medical_Keyword_15
int64:  Medical_Keyword_16
int64:  Medical_Keyword_17
int64:  Medical_Keyword_18
int64:  Medical_Keyword_19
int64:  Medical_Keyword_20
int64:  Medical_Keyword_21
int64:  Medical_Keyword_22
int64:  Medical_Keyword_23
int64:  Medical_Keyword_24
int64:  Medical_Keyword_25
int64:  Medical_Keyword_26
int64:  Medical_Keyword_27
int64:  Medical_Keyword_28
int64:  Medical_Keyword_29
int64:  Medical_Keyword_30
int64:  Medical_Keyword_31
int64:  Medical_Keyword_32
int64:  Medical_Keyword_33
int64:  Medical_Keyword_34
int64:  Medical_Keyword_35
int64:  Medical_Keyword_36
int64:  Medical_Keyword_37
int64:  Medical_Keyword_38
int64:  Medical_Keyword_39
int64:  Medical_Keyword_40
int64:  Medical_Keyword_41
```

```
int64:  Medical_Keyword_42
int64:  Medical_Keyword_43
int64:  Medical_Keyword_44
int64:  Medical_Keyword_45
int64:  Medical_Keyword_46
int64:  Medical_Keyword_47
int64:  Medical_Keyword_48
int64:  Response
```

From the description of the features given in the data section of this Kaggle competition we know which are the continuous, discrete and categorical columns. Printing the data types leads to understand that continuous and discrete features have a float64 data type, while for the categorical features the data type is int64, except for one that has an object type. While some of the columns names can be used for a better understanding of data ('Height', 'Weigth', 'BMI'), in general not knowing additional informations about most of the features won't help to undestand the relationship between variables.

In [ ]:
```
sns.histplot(df,x = 'Response')
plt.title('Class distribution across the target')
plt.tight_layout()
plt.show()
```

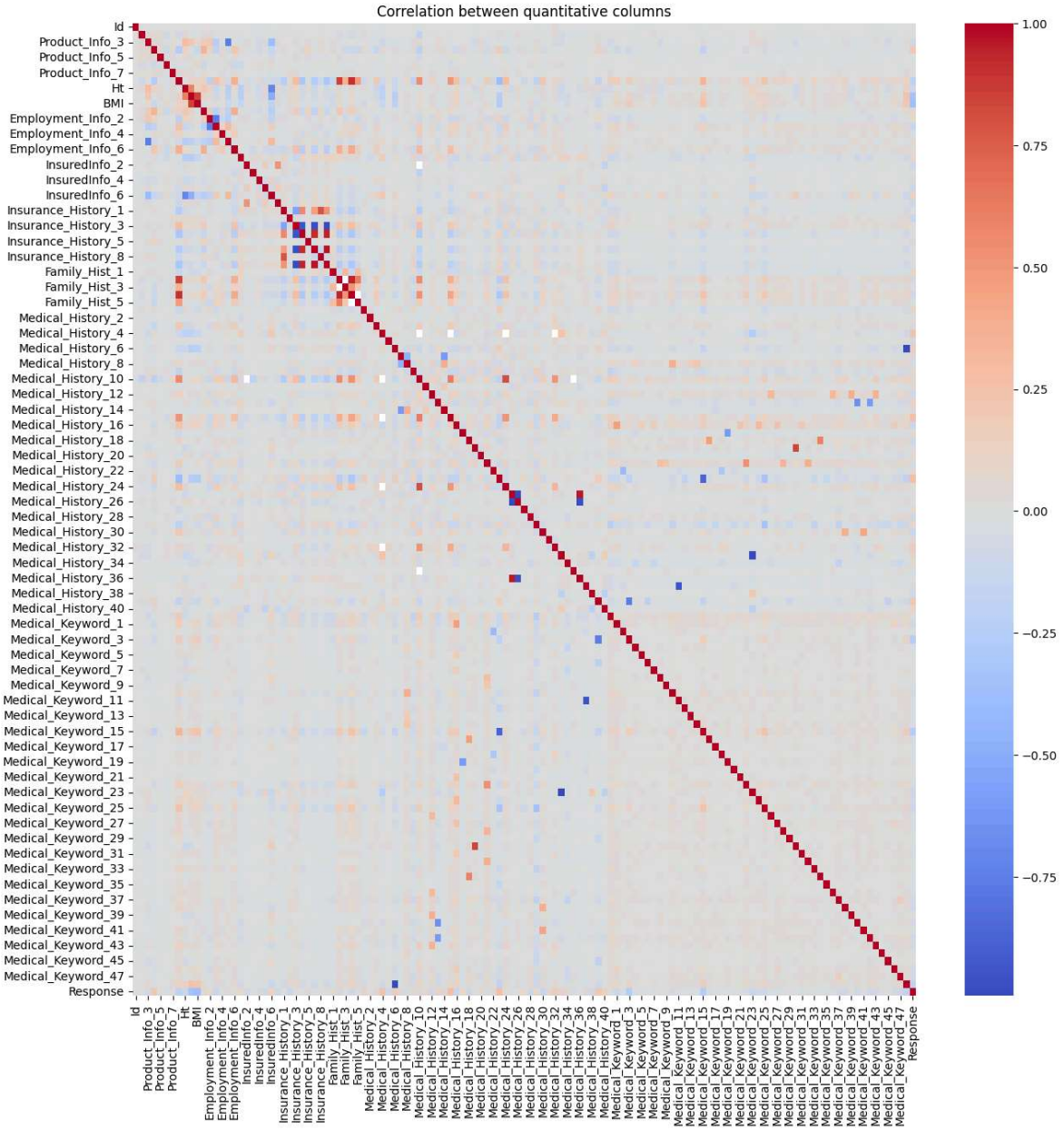
Out[ ]:
```
<function matplotlib.pyplot.plot(*args: 'float | ArrayLike | str', scalex: 'boo
l' = True, scaley: 'bool' = True, data=None, **kwargs) -> 'list[Line2D]'>
```



From the histplot above one can see that the target values distribution is unbalanced, having up to 20000 samples for the last class and less than 2000 for the third and fourth classes. This will be addressed in the modeling step, trying, where possible, to use some "class weight" parameter to balance the distribution across the classes.

In [ ]:
```
plt.figure(figsize=(15,15))
cm = df.drop(columns='Product_Info_2').corr()
plt.title('Correlation between columns')
sns.heatmap(cm, cmap = 'coolwarm')
plt.plot()
```

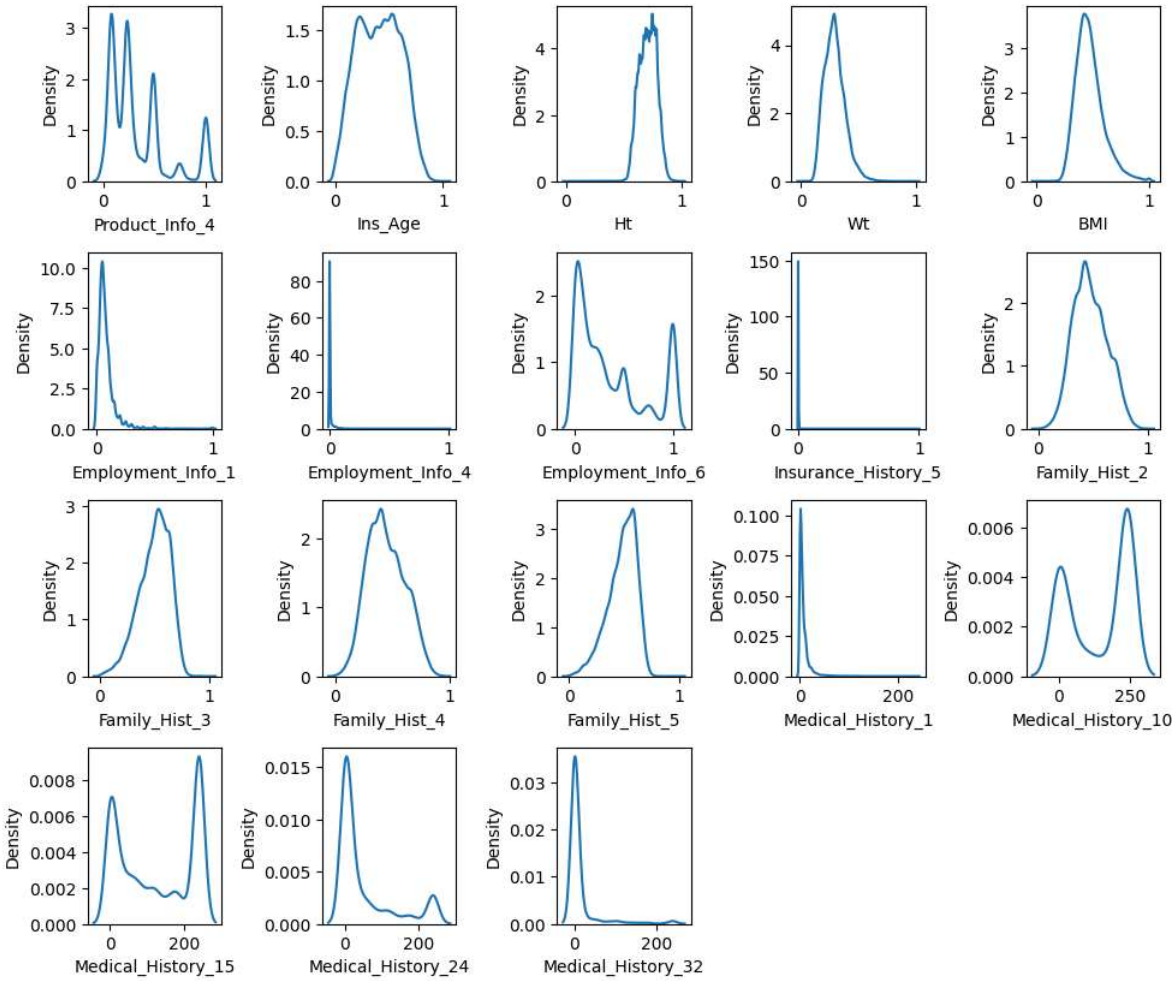
Out[ ]: []



From the correlation matrix above it can be seen that some of the features exhibit correlations between them; the reason is obvious with features like 'Height', 'Weight' or 'BMI' and for features that belong to the same group, like 'Insurance_History' or 'Family Hist' ones. Unfortunately, as said before, it is not possible to derive further informations using only their names.

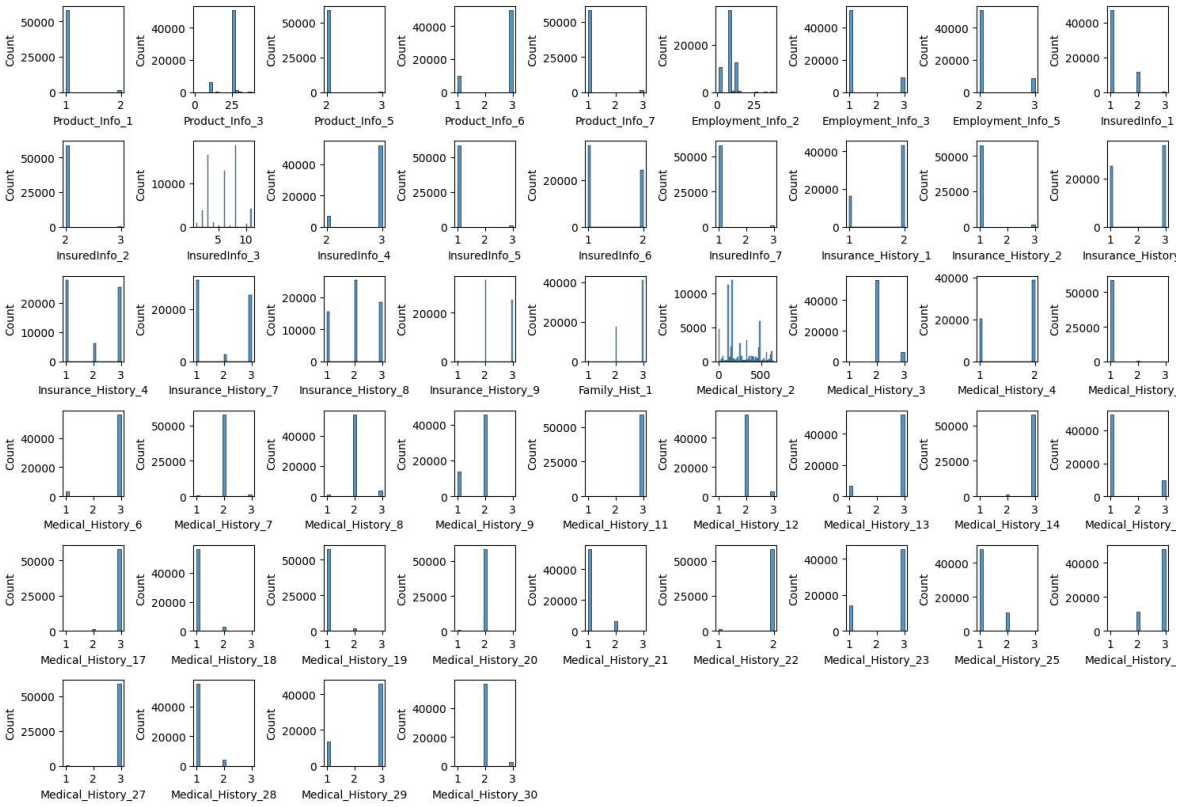
In [ ]:
```
plt.figure(figsize=(10,10))
pos =1
for col in quantitative_columns:
    plt.subplot(5, 5, pos)
    plt.xlabel(col)
    sns.kdeplot(df[col])
```

```
    pos = pos + 1
plt.tight_layout()
```



The kdeplot above is used to visualize the range and distribution of the quantitative columns. The range for the continuous features is between 0 and 1 (they're normalized) and for the discrete features is between 0 and 300. The distribution is Gaussian for some of the features, and most of them exhibit skewness.

In [ ]:
```
plt.figure(figsize=(15,15))
pos =1
for col in nominal_columns[1:50]:
    plt.subplot(9, 9, pos)
    plt.xlabel(col)
    sns.histplot(df[col])
    pos = pos + 1
plt.tight_layout()
```



In [ ]:
```
plt.figure(figsize=(15,15))
pos =1
for col in nominal_columns[50:-1]:
    plt.subplot(9, 9, pos)
    plt.xlabel(col)
    sns.histplot(df[col])
    pos = pos + 1
plt.tight_layout()
```

The same kind of analysis is done for the categorical columns; the majority of them has 3 categories, except for 'Medical_History_2' that hs more than 500, and some others that have up to 30 different categories. 'Medical_Keyword' set of variables are dummy variables, indicating the presence of certain keywords associated with the application.