In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the inpu

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output whe
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current se
```

```
/kaggle/input/stroke-prediction-dataset/healthcare-dataset-stroke-data.csv
```

İntorduction EDA for Stroke Prediction dataset

The dataset's features

1 - gender : *gender status of people*

2 - age : *age status of people*

3 - hypertension : *hypertension level status of people (normally hypertension level is 130-139 mmhg)*

4 - heart_disease : *heart_disase status of people '1' and '0'*

5 - ever_married : *married status of people **yes or no***

6 - work_type : *mode of work **self-employed or private** employee status*

7 - Residence_type: *where people live **urban or rural**.*

8 - avg_glucose_level: *avg glucose level of people(the level that has to be normally 140 mg/ dl)*

9 - bmi: *Body Mass Index level status of people*

10 - smoking_status : *smoking status for people*(**formerly or never smoked**)*

11 - stroke : *stroke status of people ('1' or '0')*

Load And Check data

In [31]:

```
data = pd.read_csv('/kaggle/input/stroke-prediction-dataset/healthcare-dataset-stroke-data.csv')

# take a information from data for datatypes , and null values
data.info()

# first five elements in data
data.head()

# control the null or unknown elements in data
data.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 5110 non-null   int64  
 1   gender             5110 non-null   object 
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64  
 4   heart_disease      5110 non-null   int64  
 5   ever_married       5110 non-null   object 
 6   work_type          5110 non-null   object 
 7   Residence_type     5110 non-null   object 
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object 
 11  stroke             5110 non-null   int64  
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

Out[31]:

```
id                   0
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                201
smoking_status       0
stroke               0
dtype: int64
```

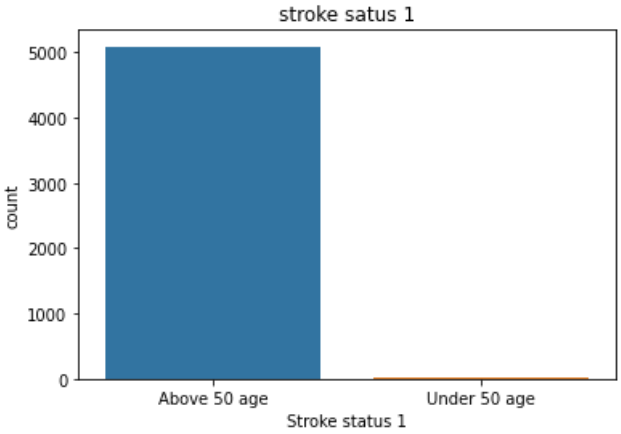
Comment The dataset's feature bmi is a lot NULL element

Visualization of the relationship between stroke and age

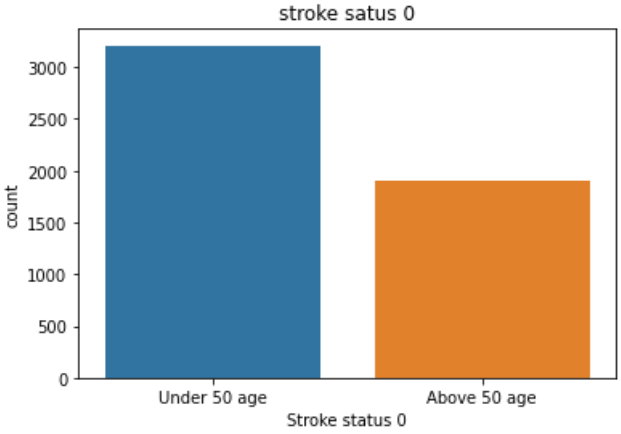
In [3]:

```
# data preparation
data_stroke_1 = data.age[data['stroke'] == 1]
data_stroke_0 = data.age[data['stroke'] == 0]

# a new data frame created for visualization
df_stroke_and_age = pd.DataFrame({'Stroke status 1' : data_stroke_1 , 'Stroke status 0' : data_stroke_0})

# NaN values are filled in a linear way
df_stroke_and_age['Stroke status 1'].interpolate('linear', inplace = True)
df_stroke_and_age['Stroke status 0'].interpolate('linear', inplace = True)

# classfication data for above 50 or under 50 age
df_stroke_and_age['Stroke status 1'] = ["Above 50 age" if i > 50 else "Under 50 age" for i in df_stroke_a
df_stroke_and_age['Stroke status 0'] = ["Above 50 age" if i > 50 else "Under 50 age" for i in df_stroke_a

# Visualization with count plot
sns.countplot(df_stroke_and_age['Stroke status 1'])
plt.title('stroke satus 1')
plt.show()

sns.countplot(df_stroke_and_age['Stroke status 0'])
plt.title('stroke satus 0')
plt.show()
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following
variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and
passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```




```
/opt/conda/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following
variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and
passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

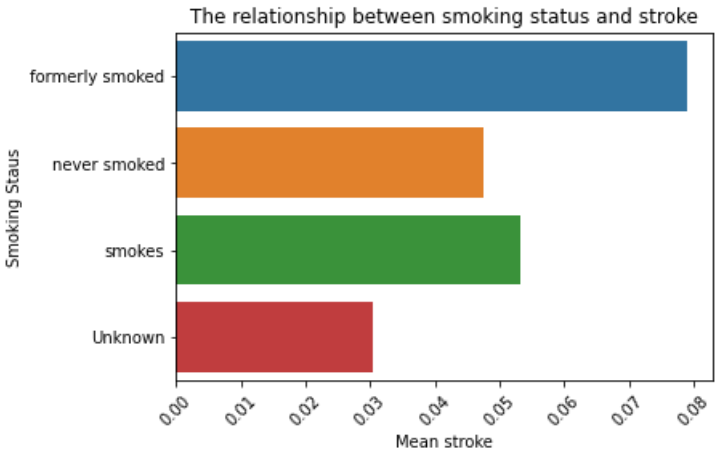


Comment : The new 50 age threshold we have created can help us make a very good classification.

The relationship between smoking status and stroke

In [4]:

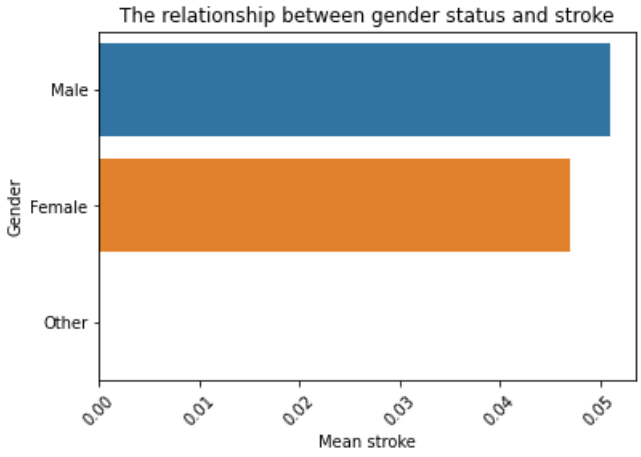
```
# provideing to datas for mean stroke by smoking status with
mean_fs_stroke = data.stroke[data['smoking_status'] == 'formerly smoked'].mean()
mean_s_stroke = data.stroke[data['smoking_status'] == 'smokes'].mean()
mean_ns_stroke = data.stroke[data['smoking_status'] == 'never smoked'].mean()
mean_uk_stroke = data.stroke[data['smoking_status'] == 'Unknown'].mean()

mean_stroke = [mean_fs_stroke , mean_ns_stroke , mean_s_stroke , mean_uk_stroke]

# smoking status
smoking_status = data['smoking_status'].unique()

sns.barplot(x = mean_stroke , y = smoking_status)
plt.title('The relationship between smoking status and stroke')
plt.xticks(rotation = 45)
plt.xlabel('Mean stroke')
plt.ylabel('Smoking Staus')
plt.show()
```



The relationship between gender and stroke

In [5]:

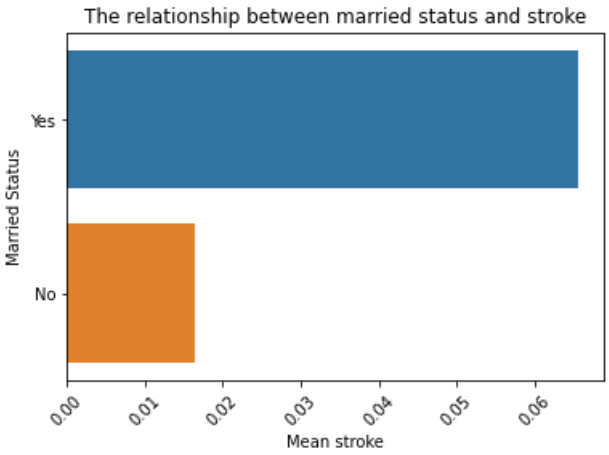
```
mean_male_stroke = data.stroke[data['gender'] == 'Male'].mean()
mean_female_stroke = data.stroke[data['gender'] == 'Female'].mean()
mean_other_stroke = data.stroke[data['gender'] == 'Other'].mean()

mean_gender = [mean_male_stroke , mean_female_stroke , mean_other_stroke]

gender_status = data['gender'].unique()

sns.barplot(x = mean_gender , y = gender_status)
plt.title('The relationship between gender status and stroke')
plt.xticks(rotation = 45)
plt.xlabel('Mean stroke')
plt.ylabel('Gender')
plt.show()
```



The relationship between married status and stroke

In [6]:

```
mean_married_stroke = data.stroke[data['ever_married'] == 'Yes'].mean()
mean_unmarried_stroke = data.stroke[data['ever_married'] == 'No'].mean()

mean_married = [mean_married_stroke , mean_unmarried_stroke]

married_status = data['ever_married'].unique()

sns.barplot(x = mean_married , y = married_status)
plt.title('The relationship between married status and stroke')
plt.xticks(rotation = 45)
plt.xlabel('Mean stroke')
plt.ylabel('Married Status')
plt.show()
```

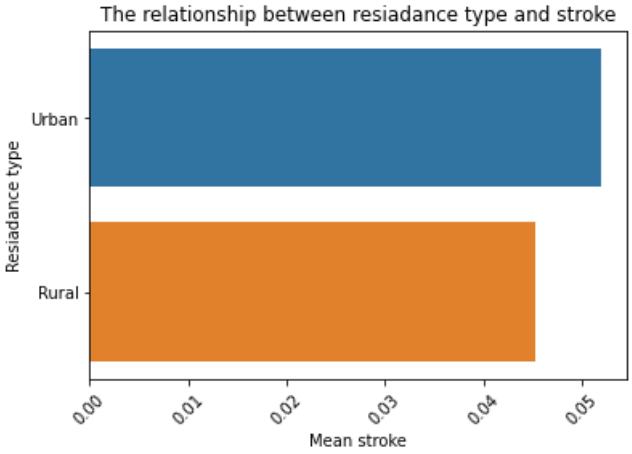


Comment : a convenient feature for classification for stroke

The relationship between residance type and stroke

In [7]:

```
mean_urban_stroke = data.stroke[data['Residence_type'] == 'Urban'].mean()
mean_rural_stroke = data.stroke[data['Residence_type'] == 'Rural'].mean()

mean_resiadance_type = [mean_urban_stroke , mean_rural_stroke]

resiadance_type = data['Residence_type'].unique()

sns.barplot(x = mean_resiadance_type, y = resiadance_type)
plt.title('The relationship between resiadance type and stroke')
plt.xticks(rotation = 45)
plt.xlabel('Mean stroke')
plt.ylabel('Resiadance type')
plt.show()
```

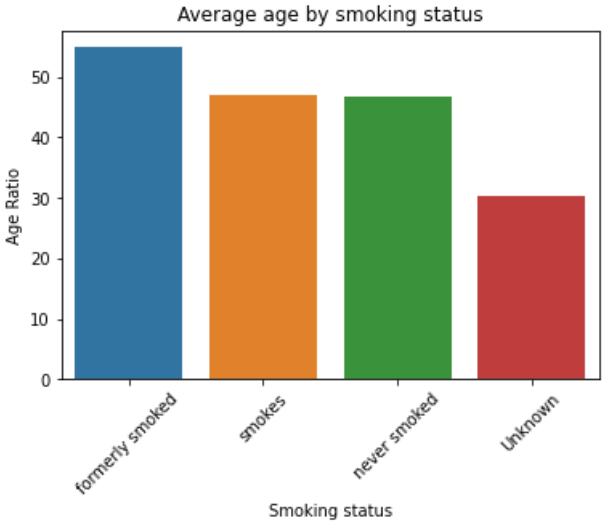


Average age by smoking status

In [8]:

```
# provideing list for average age by smoking status

age_ratios = []

# calculate to age ratios by smoking status
for i in smoking_status:
    smoke_list = data[data['smoking_status'] == i]
    age_ratio = sum(smoke_list.age) / len(smoke_list)
    age_ratios.append(age_ratio)

df_age_smoke = pd.DataFrame({'Smoking Status' : smoking_status , 'Age Ratio': age_ratios})

new_index = df_age_smoke['Age Ratio'].sort_values(ascending = False).index.values
df_age_smoke = df_age_smoke.reindex(new_index)

sns.barplot(x = 'Smoking Status', y = 'Age Ratio' , data = df_age_smoke)
plt.title('Average age by smoking status')
plt.xticks(rotation = 45)
```

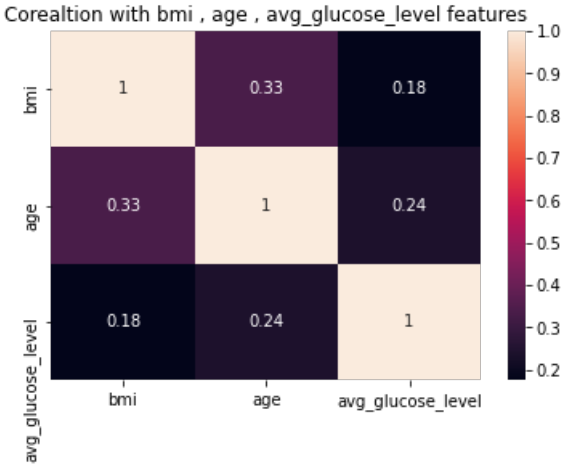
```
plt.xlabel('Smoking status')
plt.ylabel('Age Ratio')
plt.show()
```



Corealtion with bmi , age , avg_glucose_level features

In [9]:

```
# provide a new data
df = data.loc[:,['bmi' , 'age' , 'avg_glucose_level']]

sns.heatmap(df.corr() , annot = True)
plt.title('Corealtion with bmi , age , avg_glucose_level features')
plt.show()
```



Demonstration of smoking situations with pieplot

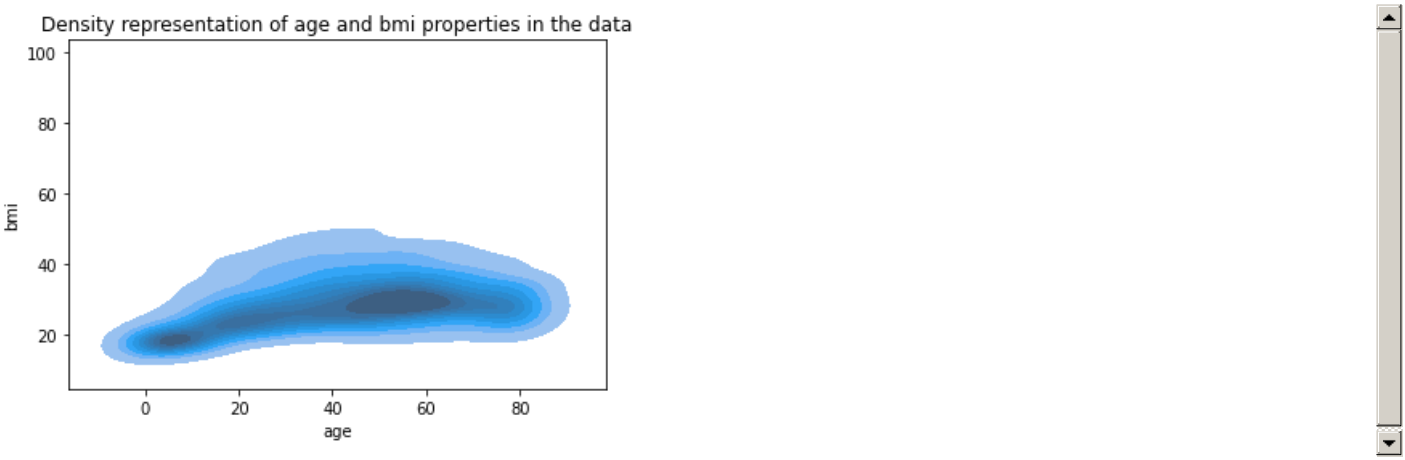
In [10]:

```
sizes = data['smoking_status'].value_counts().values

colors = ['red' , 'blue' , 'cyan' , 'yellow']

explode = [0 , 0 , 0 , 0]

labels = data['smoking_status'].value_counts().index

plt.pie(sizes , explode = explode , colors = colors , labels = labels ,  autopct='%1.1f%%')
plt.title('Demonstration of smoking situations with pieplot')
plt.show()
```

Density representation of age and bmi properties in the data

In [11]:

```
sns.kdeplot(x = 'age', y = 'bmi', data = data , shade = True)
plt.title('Density representation of age and bmi properties in the data')
plt.show()
```



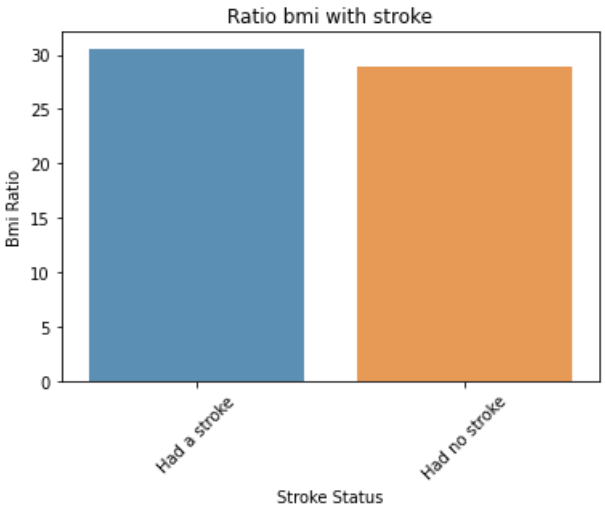
Comment: 40 and 60 years old and 20 and 40 bmi levels are the most abundant values in the data.

Ratio bmi with stroke

In [73]:

```
# provideing data for classfication

# data['stroke'] = ["Had a stroke" if i == 1 else "Had no stroke" for i in data['stroke']]

list_stroke = data['stroke'].unique()

# createing to list for memoryzation ratios
bmi_ratios = []

# filling the missing values with linear method
#data['bmi'].interpolate('linear' , inplace = True) (it has to run only once)

# calculateing to bmi ratios

for i in list_stroke:

    bmi_value = data[data['stroke'] == i]
    bmi_ratio = sum(bmi_value.bmi) / len(bmi_value.bmi)
    bmi_ratios.append(bmi_ratio)

# createing to new data frame
df_bmi = pd.DataFrame({'Stroke Status' : list_stroke , 'Bmi Ratio' : bmi_ratios})

# sorting ratio values
new_index = df_bmi['Bmi Ratio'].sort_values(ascending = False).index

# reindex to data
sorted_df_bmi = df_bmi.reindex(new_index)

# visualization ratios
```

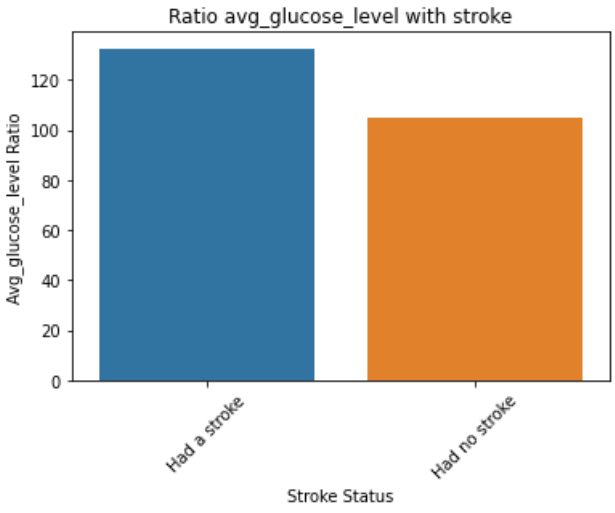
```
sns.barplot(x = 'Stroke Status' , y = 'Bmi Ratio' , data = sorted_df_bmi , alpha = 0.8)
plt.xticks(rotation = 45)
plt.title('Ratio bmi with stroke')
plt.show()
```



Ratio avg_glucose_level with stroke

In [78]:

```
# provideing data for classfication

# data['stroke'] = ["Had a stroke" if i == 1 else "Had no stroke" for i in data['stroke']]

list_stroke = data['stroke'].unique()

# createing to list for memoryzation ratios
avg_glucose_level_ratios = []

# calculateing to avg_glucose_level ratios

for i in list_stroke:

    avg_glucose_level_value = data[data['stroke'] == i]
    avg_glucose_level_ratio = sum(avg_glucose_level_value.avg_glucose_level) / len(avg_glucose_level_valu
    avg_glucose_level_ratios.append(avg_glucose_level_ratio)

# createing to new data frame
df_avg_glucose_level = pd.DataFrame({'Stroke Status' : list_stroke , 'Avg_glucose_level Ratio' : avg_gluc

# sorting ratio values
new_index = df_avg_glucose_level['Avg_glucose_level Ratio'].sort_values(ascending = False).index

# reindex to data
sorted_df_avg_glucose_level = df_avg_glucose_level.reindex(new_index)

# visualization ratios

sns.barplot(x = 'Stroke Status' , y = 'Avg_glucose_level Ratio' , data = sorted_df_avg_glucose_level)
plt.xticks(rotation = 45)
plt.title('Ratio avg_glucose_level with stroke')
plt.show()
```



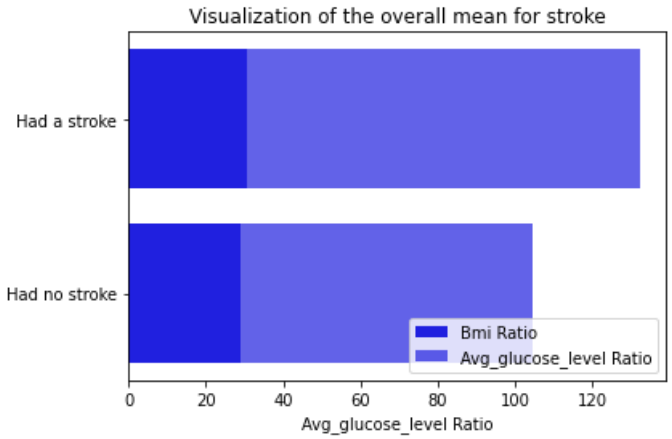
Visualization of the overall mean for stroke

In [99]:

```
sns.barplot(x = 'Bmi Ratio' , y = list_stroke , data = sorted_df_bmi , color = 'blue' , label = 'Bmi Ratio
sns.barplot(x = 'Avg_glucose_level Ratio' , y = list_stroke , data = sorted_df_avg_glucose_level , color =
plt.legend(loc='lower right',frameon = True)
plt.title('Visualization of the overall mean for stroke')
```

Out[99]:

```
Text(0.5, 1.0, 'Visualization of the overall mean for stroke')
```



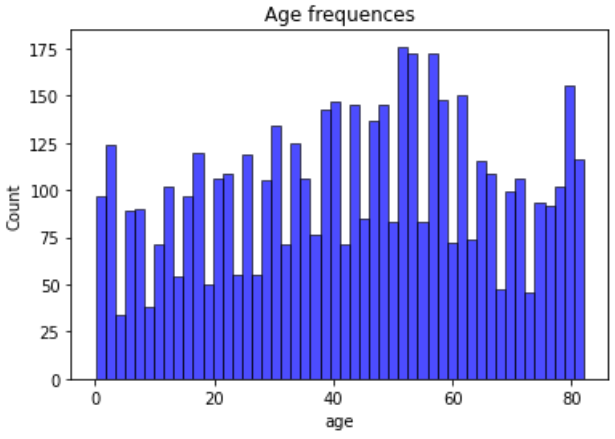
Calculating frequencies for numerical categories

In [105]:

```
sns.histplot(data['age'] , bins = 50 , color = 'blue' , alpha = 0.7)
plt.title('Age frequences')
```

Out[105]:

```
Text(0.5, 1.0, 'Age frequences')
```

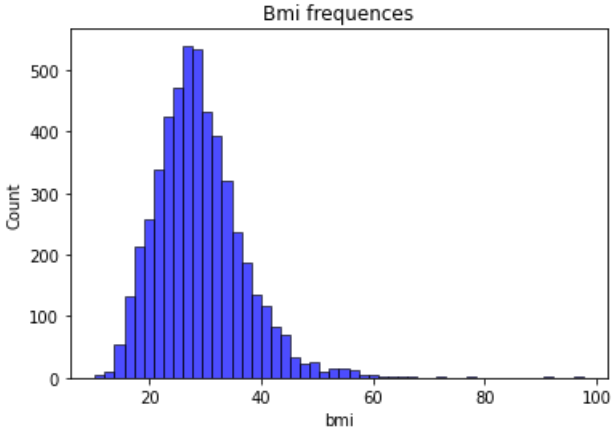


In [106]:

```
sns.histplot(data['bmi'] , bins = 50 , color = 'blue' , alpha = 0.7)
plt.title('Bmi frequences')
```

Out[106]:

```
Text(0.5, 1.0, 'Bmi frequences')
```



In [107]:

```
sns.histplot(data['avg_glucose_level'] , bins = 50 , color = 'blue' , alpha = 0.7)
plt.title('Avg_glucose_level frequences')
```

Out[107]:

```
Text(0.5, 1.0, 'Avg_glucose_level frequences')
```