In [ ]:
```python
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import chi2_contingency
from scipy import stats
from math import sqrt
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeRegressor
%matplotlib inline
```

In [ ]:
```python
# Create dataframe
df = pd.read_csv("..\\..\\data\\healthcare-dataset-stroke-data.csv")
# change to // if mac
```

In [ ]:
```python
# show first five rows to get an overview
df.head()
```

Out[ ]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

In [ ]:
```python
# Desciption of each coloumn
df.describe().round(2)
```

Out[ ]:

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.00	5110.00	5110.0	5110.00	5110.00	4909.00	5110.00
mean	36517.83	43.23	0.1	0.05	106.15	28.89	0.05
std	21161.72	22.61	0.3	0.23	45.28	7.85	0.22
min	67.00	0.08	0.0	0.00	55.12	10.30	0.00
25%	17741.25	25.00	0.0	0.00	77.24	23.50	0.00
50%	36932.00	45.00	0.0	0.00	91.88	28.10	0.00
75%	54682.00	61.00	0.0	0.00	114.09	33.10	0.00
max	72940.00	82.00	1.0	1.00	271.74	97.60	1.00

In [ ]:
```python
# Attribute overview
df.info()
```

```
<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
```

In [ ]:
```python
df["hypertension"].sum()
```

Out[ ]: 498

In [ ]:
```python
df["heart_disease"].sum()
```

Out[ ]: 276

In [ ]:
```python
#BMI adult
df[df["age"]>18].bmi.describe()
```

Out[ ]:
```
count    4014.000000
mean       30.493921
std         7.222288
min        11.300000
25%        25.500000
50%        29.300000
75%        34.200000
max        92.000000
Name: bmi, dtype: float64
```

Nominal values

In [ ]:
```
df["smoking_status"].unique()
```

Out[ ]:
```
array(['formerly smoked', 'never smoked', 'smokes', 'Unknown'],
      dtype=object)
```

In [ ]:
```
df["work_type"].unique()
```

Out[ ]:
```
array(['Private', 'Self-employed', 'Govt_job', 'children', 'Never_worked'],
      dtype=object)
```

In [ ]:
```
df["Residence_type"].unique()
```

Out[ ]:
```
array(['Urban', 'Rural'], dtype=object)
```

drop irrelevant columns

In [ ]:
```
# We drop the id column, since we know we are not going to use it
df = df.drop('id',1)
```

Missing values

We want to deal with our missing values before moving on.

In [ ]:
```
# Number of null (missing) values
df.isna().sum()
```

Out[ ]:
```
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
```

In [ ]:
```
# Figuring out how many of the strokes that have a missing bmi
stroke_df = df[df['stroke'] == 1]
stroke_df.isnull().sum()
```

Out[ ]:
```
gender                0
age                   0
hypertension          0
heart_disease         0
ever_married          0
work_type             0
Residence_type        0
avg_glucose_level     0
bmi                  40
smoking_status        0
stroke                0
dtype: int64
```

imputing misssing values

In [ ]:
```
print("Shape before DecisionTreeRegressor " + str(df.shape))
```

```
Shape before DecisionTreeRegressor (5110, 11)
```

In [ ]:
```
# We create a pipeline in order to reuse it later
# We want to use a decisiontree so that we do not need to replace 1/5 of the strokes with the same value of bmi
# by replacing it with mean or median

DecisionTreePip = Pipeline(steps=[
                               ('Scale',StandardScaler()),
                               ('DecisionTreeReg',DecisionTreeRegressor(random_state = 42))
                              ])

X = df[['age','gender','bmi']]
X.gender = X.gender.replace({'Male' : 0, 'Female' : 1 , 'Other' : -1}).astype(np.uint8)

# create a dataframe containing the missing values of X
missing = X[X.bmi.isna()]

# remove the missing values from X
X = X.dropna()

# creates Y by removing bmi from X
Y = X.pop('bmi')

# fit the pipeline
DecisionTreePip.fit(X,Y)

# make the prediction
predict_bmi = pd.Series(DecisionTreePip.predict(missing[['age', 'gender']]), index = missing.index)
df.loc[missing.index, 'bmi'] = predict_bmi
```

```
C:\Users\naja\anaconda3\lib\site-packages\pandas\core\generic.py:5168: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self[name] = value
```

In [ ]:
```
print("Shape after DecisionTreeRegressor " + str(df.shape))
```

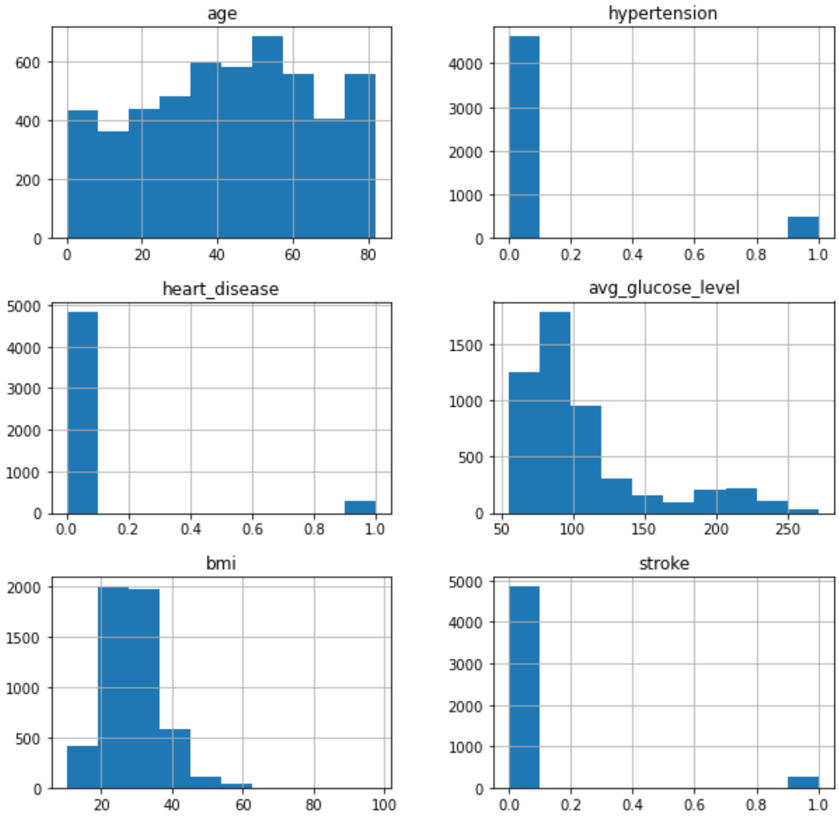
```
Shape after DecisionTreeRegressor (5110, 11)
```

In [ ]:
```
df.isnull().sum()
```

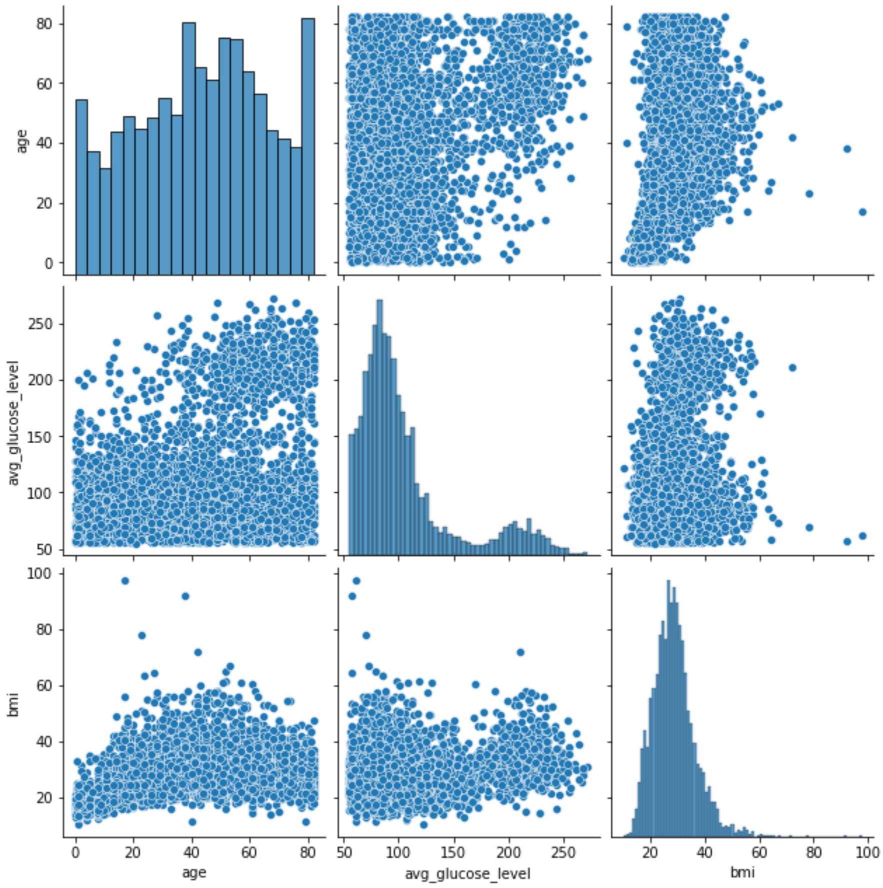
Out[ ]:
```
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                  0
smoking_status       0
stroke               0
dtype: int64
```

Visuals

In [ ]:
```
# Histograms
hist = df.hist(figsize = (10,10))
```



In [ ]:
```
# Pairplots to show where there might be clusters
sns_plot = sns.pairplot(df, height = 3, vars = ['age','avg_glucose_level', 'bmi'])
# sns_plot.savefig('pairplot.png') # saves it as a picture
plt.show()
```



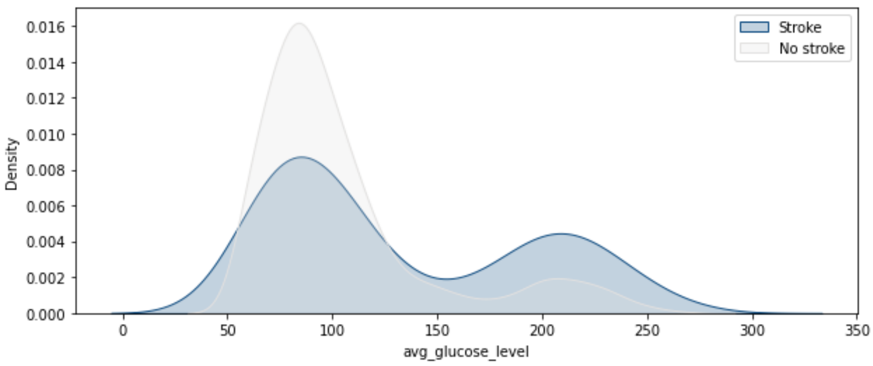
In [ ]:
```
df.select_dtypes(include=['number']).columns
```

Out[ ]:
```
Index(['age', 'hypertension', 'heart_disease', 'avg_glucose_level', 'bmi',
       'stroke'],
      dtype='object')
```

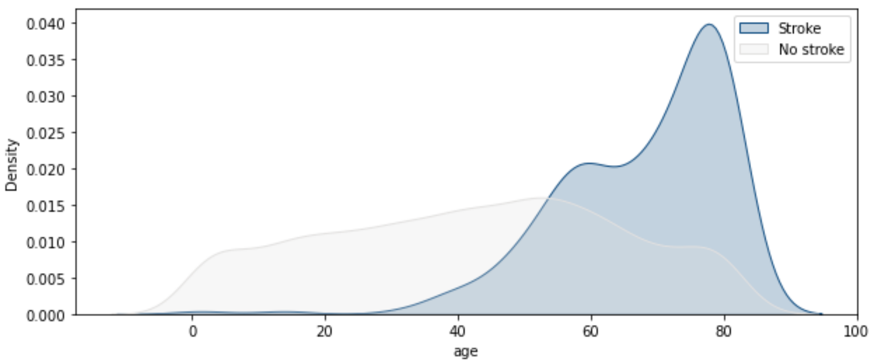
densitity plots

In [ ]:
```
#https://www.kaggle.com/joshuaswords/predicting-a-stroke-shap-lime-explainer-eli5
```

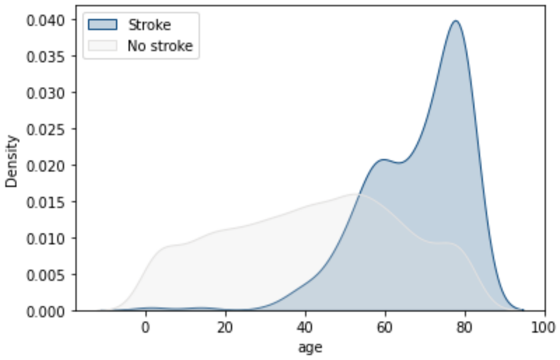
In [ ]:
```
fig=plt.figure(figsize=(10,4),facecolor='white')
fig = sns.kdeplot(data=df[df.stroke==1],x='avg_glucose_level',shade=True,color='#0f4c8180', label="Stroke")
fig = sns.kdeplot(data=df[df.stroke==0],x='avg_glucose_level',shade=True,color='#e3e2e160',label="No stroke")
#fig.get_yaxis().set_visible(False)
fig.legend()
fig.get_figure().savefig("gluc_kde_dist.png")
```



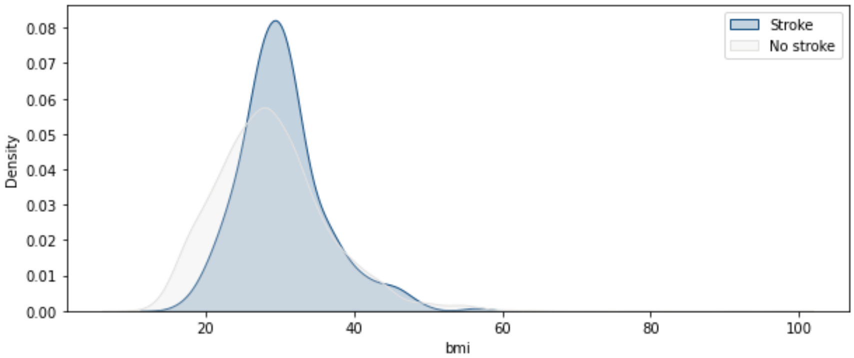
In [ ]:
```
fig=plt.figure(figsize=(10,4),facecolor='white')
fig = sns.kdeplot(data=df[df.stroke==1],x='age',shade=True,color='#0f4c81', label="Stroke")
fig = sns.kdeplot(data=df[df.stroke==0],x='age',shade=True,color='#e3e2e1',label="No stroke")
#fig.get_yaxis().set_visible(False)
fig.legend()
fig.get_figure().savefig("age_kde_dist.png")
```



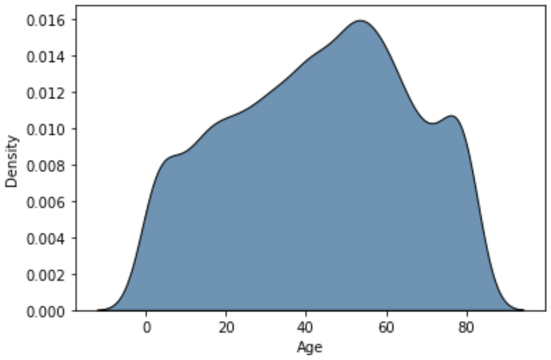
In [ ]:
```
#smaller version for report
fig=plt.figure(facecolor='white')
fig = sns.kdeplot(data=df[df.stroke==1],x='age',shade=True,color='#0f4c81', label="Stroke")
fig = sns.kdeplot(data=df[df.stroke==0],x='age',shade=True,color='#e3e2e1',label="No stroke")
#fig.get_yaxis().set_visible(False)
fig.legend(loc='upper left')
fig.get_figure().savefig("age_kde_dist_smaller.png")
```



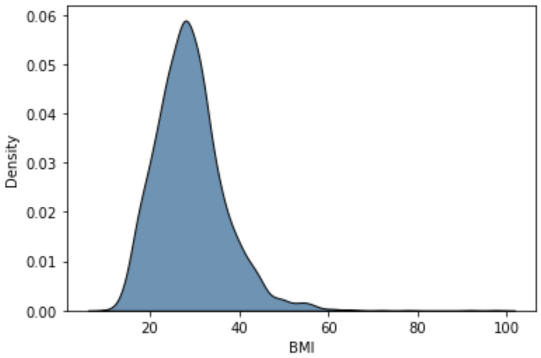
In [ ]:
```
fig=plt.figure(figsize=(10,4),facecolor='white')
fig = sns.kdeplot(data=df[df.stroke==1],x='bmi',shade=True,color='#0f4c81', label="Stroke")
fig = sns.kdeplot(data=df[df.stroke==0],x='bmi',shade=True,color='#e3e2e160',label="No stroke")
#fig.get_yaxis().set_visible(False)
fig.legend()
fig.get_figure().savefig("bmi_kde_dist.png")
```



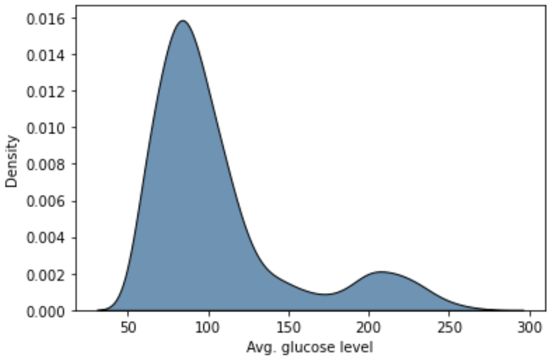
In [ ]:
```
fig = sns.kdeplot(df["age"], color='#0f4c81', shade=True, ec='black',alpha=0.6)
fig.set_xlabel('Age')
#fig.get_yaxis().set_visible(False)
fig.get_figure().savefig("age_dist.png")
```



In [ ]:
```
fig = sns.kdeplot(df["bmi"], color='#0f4c81', shade=True, ec='black',alpha=0.6)
fig.set_xlabel('BMI')
#fig.get_yaxis().set_visible(False)
fig.get_figure().savefig("bmi_dist.png")
```



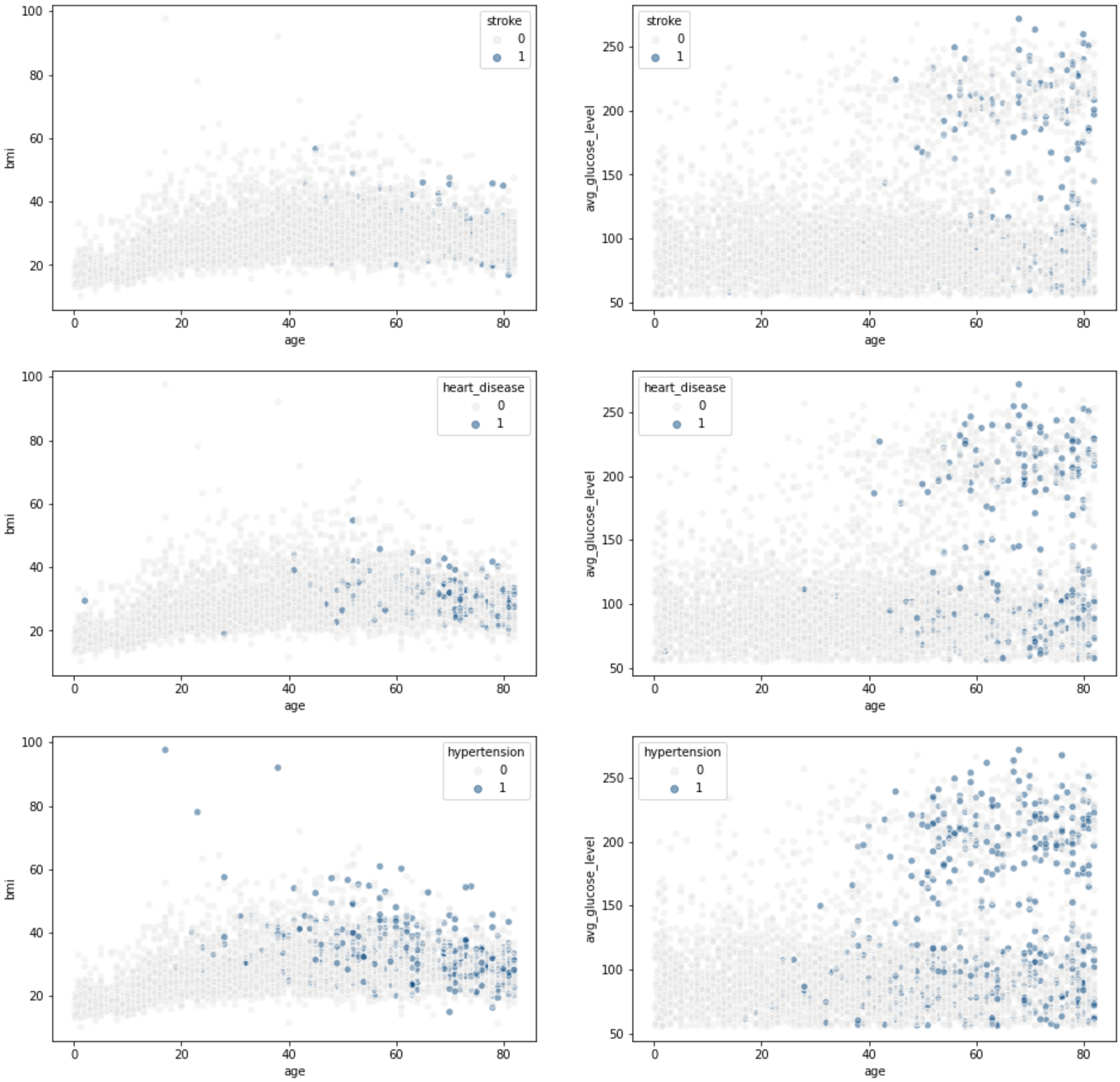
In [ ]:
```
fig = sns.kdeplot(df["avg_glucose_level"], color='#0f4c81', shade=True, ec='black',alpha=0.6)
fig.set_xlabel('Avg. glucose level')
#fig.get_yaxis().set_visible(False)
fig.get_figure().savefig("avg_gluc_dist.png")
```



age and ..

In [ ]:

```python
# Scatterplots on two-variable relationships between age(discrete) and continuous variables
# and binary value (stroke, heart disease and hypertension)

fig, axs = plt.subplots(3,2, figsize=(16,16))

color_dict = dict({1:"#0f4c8180",0:"#e3e2e160"})

x=df["age"]

sns.scatterplot(ax = axs[0,0], x=x, y=df["bmi"], hue=df["stroke"], palette=color_dict)
sns.scatterplot(ax = axs[0,1], x=x, y=df["avg_glucose_level"], hue=df["stroke"],palette=color_dict)
sns.scatterplot(ax = axs[1,0], x=x, y=df["bmi"], hue=df["heart_disease"],palette=color_dict)
sns.scatterplot(ax = axs[1,1], x=x, y=df["avg_glucose_level"], hue=df["heart_disease"],palette=color_dict)
sns.scatterplot(ax = axs[2,0], x=x, y=df["bmi"], hue=df["hypertension"],palette=color_dict)
sns.scatterplot(ax = axs[2,1], x=x, y=df["avg_glucose_level"], hue=df["hypertension"],palette=color_dict)

fig.savefig("scatterplot_dist.png")
```



Overall:

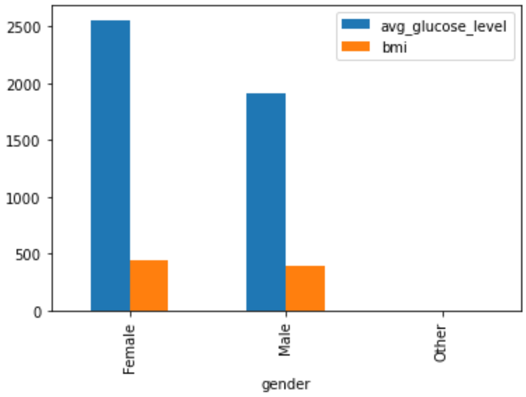
- The higher the age, the more have problems in general (strokes, heart dieseases, hypertension)

- The higher the glucose level, the more seem to have strokes at an higher age!

Strokes:

- High age + high glucose level -> a lot of strokes
- High age but high BMI -> not so many strokes Heart disease:
- The higher the age, the more heart diseases
- But high BMI does not seem to have a great influence
- Avg glucose level does not seem to matter a lot too Hypertension
- Also, the higher the age, the more have hypertension, but definitely less clear than for other two
- Interestingly, all outliers with very high BMI have also hypertension
- Avg glucose level really does not influence it much

In [ ]:
```
# BMI and glucose level within men and women
df.groupby(['gender']).nunique().plot(kind = 'bar', y = ['avg_glucose_level', 'bmi'])
plt.show()
```

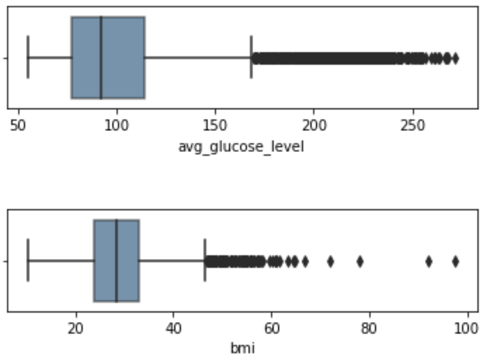


Outliers

In [ ]:
```
df.select_dtypes(include=['number']).columns
```

Out[ ]:
```
Index(['age', 'hypertension', 'heart_disease', 'avg_glucose_level', 'bmi',
       'stroke'],
      dtype='object')
```

In [ ]:
```
fig, axs = plt.subplots(2)
plt.subplots_adjust(hspace = 1)
col = ['avg_glucose_level','bmi']

for i, col in enumerate(col):
    fig = sns.boxplot(x = df[col],ax=axs[i], color='#0f4c81', boxprops=dict(alpha=.6))

fig.get_figure().savefig("outliers.png")
```



In [ ]:
```
# Outliers in BMI
#fig = sns.boxplot(x = df['bmi'],color='#0f4c81',boxprops=dict(alpha=.6))
#fig.get_figure().savefig("bmi_outliers.png")
```

In [ ]:
```
# Outliers on glukose level
#fig = sns.boxplot(x = df['avg_glucose_level'])
```

In [ ]:
```
# Outliers on age
#fig = sns.boxplot(x = df['age'])
```

Stroke in different groups

In [ ]:
```
# Strokes within men and women
print((df[df["gender"]=="Male"].sum()["stroke"]/df[df["gender"]=="Male"].count()["stroke"])*100)
print((df[df["gender"]=="Female"].sum()["stroke"]/df[df["gender"]=="Female"].count()["stroke"])*100)
```

```
5.106382978723404
4.709418837675351
```

In [ ]:
```
# Strokes divided between urban and rural
print((df[df["Residence_type"]=="Urban"].sum()["stroke"]/df[df["Residence_type"]=="Urban"].count()["stroke"])*100)
```

```
print((df[df["Residence_type"]=="Rural"].sum()["stroke"]/df[df["Residence_type"]=="Rural"].count()["stroke"])*100)
```

```
5.200308166409862
4.534606205250596
```

In [ ]:
```
# Percent of people who have a heartdisease and a stroke, and people who had a stroke but no heart disease
print((df[df["heart_disease"]==1].sum()["stroke"]/df[df["heart_disease"]==1].count()["stroke"])*100)
print((df[df["heart_disease"]==0].sum()["stroke"]/df[df["heart_disease"]==0].count()["stroke"])*100)
```

```
17.02898550724638
4.178733967728589
```

In [ ]:
```
# Percent of people who have a hypertension and a stroke, and people who had a stroke but no hypertension
print((df[df["hypertension"]==1].sum()["stroke"]/df[df["hypertension"]==1].count()["stroke"])*100)
print((df[df["hypertension"]==0].sum()["stroke"]/df[df["hypertension"]==0].count()["stroke"])*100)
```

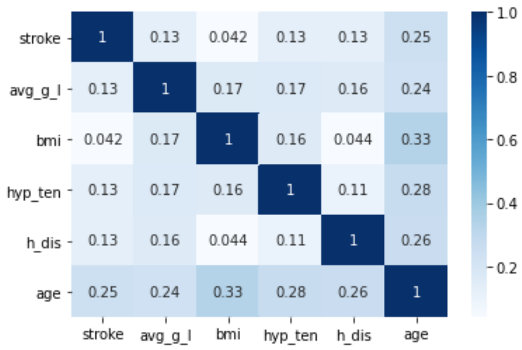
```
13.253012048192772
3.967909800520382
```

In [ ]:
```
# Smoking status and stroke
print((df[df["smoking_status"]=="never smoked"].sum()["stroke"]/df[df["smoking_status"]=="never smoked"].count()["stroke"])*100)
print((df[df["smoking_status"]=="Unknown"].sum()["stroke"]/df[df["smoking_status"]=="Unknown"].count()["stroke"])*100)
print((df[df["smoking_status"]=="formerly smoked"].sum()["stroke"]/df[df["smoking_status"]=="formerly smoked"].count()["stroke"])*100)
print((df[df["smoking_status"]=="smokes"].sum()["stroke"]/df[df["smoking_status"]=="smokes"].count()["stroke"])*100)
```

```
4.7568710359408035
3.0440414507772022
7.909604519774012
5.323193916349809
```

Correlation

In [ ]:
```
# Correlation heatmap (only between two variables)

numerical = df[["stroke", "avg_glucose_level", "bmi", "hypertension", "heart_disease", "age"]]
numerical = numerical.rename(columns={"avg_glucose_level": "avg_g_l", "heart_disease": "h_dis","hypertension":"hyp_ten"})
fig = sns.heatmap(numerical.corr(), annot=True,cmap="Blues")
fig.get_figure().savefig("heatmap.png")

#We can't see a strong positive correlation between any numerical value and a stroke
#The largest positive correlation for a stroke is with age
```



Chi square test

Exploring the relationship between categorical values by using a Chi Square Test

In [ ]:
```
#First look at smoking_status -> stroke!

# Determine significance level
significance_level = 0.05

chisqt = pd.crosstab(df.smoking_status, df.stroke, margins=True)
value = np.array([chisqt.iloc[0][0:5].values,
                  chisqt.iloc[1][0:5].values])
print(chi2_contingency(value)[0:3])
# first value chi-square, second value p-value, then degrees of freedom
if chi2_contingency(value)[1] <= significance_level:
    print("The two varibles have a significant correlation!")
else:
    print("The two varibles have NO significant correlation!")
# chisqt
```

```
(29.047623229232553, 4.924801830448134e-07, 2)
The two varibles have a significant correlation!
```

Now want to determine how much a category value contributes to stroke,

therefore we conduct post hoc testing. Conduct multiple 2×2 Chi-square tests using the Bonferroni-adjusted p-value.

In [ ]:
```
# Determine the Bonferroni-adjusted p-value
# The formula is p/N, where “p”= the original tests p-value and “N”= the number of planned pairwise comparisons
bonferroni_p = 0.05/3

dummies = pd.get_dummies(df['smoking_status'])
dummies.drop(["Unknown"], axis= 1, inplace= True)
dummies.head()
```

Out[ ]:

	formerly smoked	never smoked	smokes
0	1	0	0

	formerly smoked	never smoked	smokes
1	0	1	0
2	0	1	0
3	0	0	1
4	0	1	0

In [ ]:

```
# Check whether they are significant and also calculate phi coefficient (like pearson)
# https://www.statisticshowto.com/phi-coefficient-mean-square-contingency-coefficient/
# https://en.wikipedia.org/wiki/Phi_coefficient

for series in dummies:
    nl = "\n"
    crosstab = pd.crosstab(dummies[f"{series}"], df['stroke'])
    print(crosstab, nl)
    a = crosstab.loc[0][0] # needed for phi coefficient
    b = crosstab.loc[0][1]
    c = crosstab.loc[1][0]
    d = crosstab.loc[1][1]
    phi_coeff = (a*d-b*c)/sqrt((a+b)*(c+d)*(a+c)*(b+d))
    chi2, p, dof, expected = stats.chi2_contingency(crosstab)
    significance = p <= bonferroni_p
    print(f"Chi2 value= {chi2}{nl}p-value= {p}{nl}Degrees of freedom= {dof}{nl}Significant {significance}{nl}Phi coefficient {phi_coeff}")
    print()
```

```
stroke              0    1
formerly smoked
0                4046  179
1                 815   70

Chi2 value= 20.510528545500733
p-value= 5.930414820659066e-06
Degrees of freedom= 1
Significant True
Phi coefficient 0.06455557529819718

stroke           0    1
never smoked
0             3059  159
1             1802   90

Chi2 value= 0.05191850370394257
p-value= 0.8197579391268933
Degrees of freedom= 1
Significant False
Phi coefficient -0.004128687879042821

stroke     0    1
smokes
0       4114  207
1        747   42

Chi2 value= 0.3015133498043404
p-value= 0.5829352382079063
Degrees of freedom= 1
Significant False
Phi coefficient 0.008939203206269937
```

Therefore only the category value "formerly smoked" is significant

-> Means that a higher proportion of people who had a stroke also smoked formerly!

-> But VERY low phi coefficient, therefore no true correlation (Note that correlation does not imply causality.

That is, if A and B are correlated, this does not necessarily imply that A causes B or that B causes)

In [ ]:

```
# Next look at work_type -> stroke!
# Determine significance level
significance_level = 0.05

chisqt = pd.crosstab(df.work_type, df.stroke, margins=True)
value = np.array([chisqt.iloc[0][0:5].values,
                  chisqt.iloc[1][0:5].values])
print(chi2_contingency(value)[0:3])
# first value chi-square, second value p-value, then degrees of freedom
if chi2_contingency(value)[1] <= significance_level:
    print("The two varibles have a significant correlation!")
else:
    print("The two varibles have NO significant correlation!")
# chisqt
```

```
(1.1614713657633202, 0.5594866104012024, 2)
The two varibles have NO significant correlation!
```

In [ ]:

```
# Next look at residence type -> stroke!
# Determine significance level
significance_level = 0.05

chisqt = pd.crosstab(df.Residence_type, df.stroke, margins=True)
value = np.array([chisqt.iloc[0][0:5].values,
                  chisqt.iloc[1][0:5].values])
print(chi2_contingency(value)[0:3])
# first value chi-square, second value p-value, then degrees of freedom
if chi2_contingency(value)[1] <= significance_level:
    print("The two varibles have a significant correlation!")
else:
```

```
    print("The two varibles have NO significant correlation!")
# chisqt
```

```
(1.2210278401168941, 0.5430717019050562, 2)
The two varibles have NO significant correlation!
```

In [ ]:

```
# Next look at marriage -> stroke!
# Determine significance level
significance_level = 0.05

chisqt = pd.crosstab(df.ever_married, df.stroke, margins=True)
value = np.array([chisqt.iloc[0][0:5].values,
                  chisqt.iloc[1][0:5].values])
print(chi2_contingency(value)[0:3])
# first value chi-square, second value p-value, then degrees of freedom
if chi2_contingency(value)[1] <= significance_level:
    print("The two varibles have a significant correlation!")
else:
    print("The two varibles have NO significant correlation!")
#chisqt
```

```
(59.978623069992466, 9.458178026759999e-14, 2)
The two varibles have a significant correlation!
```

In [ ]:

```
dummies = pd.get_dummies(df['ever_married'])
# dummies.drop(["Unknown"], axis= 1, inplace= True)
```

In [ ]:

```
# Check whether they are significant and also calculate phi coefficient (like pearson)
# https://www.statisticshowto.com/phi-coefficient-mean-square-contingency-coefficient/
# https://en.wikipedia.org/wiki/Phi_coefficient

for series in dummies:
    nl = "\n"
    crosstab = pd.crosstab(dummies[f"{series}"], df['stroke'])
    print(crosstab, nl)
    a = crosstab.loc[0][0] # needed for phi coefficient
    b = crosstab.loc[0][1]
    c = crosstab.loc[1][0]
    d = crosstab.loc[1][1]
    phi_coeff = (a*d-b*c)/sqrt((a+b)*(c+d)*(a+c)*(b+d))
    chi2, p, dof, expected = stats.chi2_contingency(crosstab)
    significance = p <= bonferroni_p
    print(f"Chi2 value= {chi2}{nl}p-value= {p}{nl}Degrees of freedom= {dof}{nl}Significant {significance}{nl}Phi coefficient {phi_coeff}")
    print()
```

```
stroke     0    1
No
0       3133  220
1       1728   29

Chi2 value= 58.923890259034195
p-value= 1.6389021142314745e-14
Degrees of freedom= 1
Significant True
Phi coefficient -0.10833974165701027

stroke     0    1
Yes
0       1728   29
1       3133  220

Chi2 value= 58.923890259034195
p-value= 1.6389021142314745e-14
Degrees of freedom= 1
Significant True
Phi coefficient 0.10833974165701027
```