A stroke occurs when blood supply is blocked or a blood vessel bursts in the brain. Strokes can cause brain damage, long-term disabilities and in the most severe cases, death. According to the Centers for Disease Control and Prevention, someone in the United States will suffer from a stroke every forty seconds. Along with cancer and heart disease, strokes are a leading cause of death for Americans. There are a wide variety of factors that can contribute to a person's likelihood of suffering from a stroke. Some risk factors include smoking, obesity, age and high blood pressure (Centers for Disease Control and Prevention [CDC], 2021). Stroke prediction is essential in saving lives. Therefore, it is important that we understand the causes of a stroke in order to make better recommendations to those at risk through the utilization of predictive modeling.

In [1]:
```
from pathlib import Path

import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
import matplotlib.pylab as plt
from sklearn import preprocessing
from dmba import plotDecisionTree, classificationSummary, regressionSummary
from sklearn.linear_model import LinearRegression, Lasso, Ridge, LassoCV, BayesianRidge, LogisticRegressionC
import statsmodels.formula.api as sm
import matplotlib.pylab as plt
import matplotlib as mpl
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import seaborn as sns
from dmba import gainsChart
from dmba import regressionSummary, exhaustive_search
from dmba import backward_elimination, forward_selection, stepwise_selection
from dmba import adjusted_r2_score, AIC_score, BIC_score
from sklearn.metrics import classification_report
from sklearn.naive_bayes import MultinomialNB
```

In [2]:
```
stroke_df = pd.read_csv(r'C:\Users\ahzha\Desktop\stroke_data.csv', header = 0, sep = ',')
```

Inspecting the raw data is an important step in optimizing the best models. We begin our exploratory data analysis by importing relevant packages and the chosen dataset into the environment. Following this step, the structure of the dataset is acquired.

In [3]:
```
stroke_df.shape
```

Out[3]:
```
(5110, 12)
```

In [4]:
```
stroke_df.columns
```

Out[4]:
```
Index(['id', 'gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
       'work_type', 'Residence_type', 'avg_glucose_level', 'bmi',
       'smoking_status', 'stroke'],
      dtype='object')
```

Inspecting statistical measures for numerical variables allows us to detect outliers and handle missing data.

In [5]:
```
stroke_df.describe() #summary statistics for the numerical variables
```

Out[5]:

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	4909.000000	5110.000000
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.893237	0.048728
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.854067	0.215320
min	67.000000	0.080000	0.000000	0.000000	55.120000	10.300000	0.000000
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.500000	0.000000
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.100000	0.000000
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	33.100000	0.000000
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.600000	1.000000

In [6]:
```
stroke_df.dtypes # check the data types
```

Out[6]:
```
id                     int64
gender                object
age                  float64
hypertension           int64
heart_disease          int64
ever_married          object
work_type             object
Residence_type        object
avg_glucose_level    float64
bmi                  float64
smoking_status        object
stroke                 int64
dtype: object
```

In [7]:
```
counts = stroke_df.nunique() #check for unique counts. need to check gender for the three different counts,
counts
```

Out[7]:
```
id                   5110
gender                  3
age                   104
hypertension            2
heart_disease           2
ever_married            2
work_type               5
Residence_type          2
avg_glucose_level    3979
bmi                   418
smoking_status          4
stroke                  2
dtype: int64
```

In [8]:
```
stroke_df['gender'].value_counts()# checking gender categories
```

Out[8]:
```
Female    2994
Male      2115
Other        1
Name: gender, dtype: int64
```

In [9]:
```
stroke_df['work_type'].value_counts()
```

Out[9]:
```
Private          2925
Self-employed     819
children          687
Govt_job          657
Never_worked       22
Name: work_type, dtype: int64
```

In [10]:
```
stroke_df['smoking_status'].value_counts()
```

Out[10]:
```
never smoked       1892
Unknown            1544
formerly smoked     885
smokes              789
Name: smoking_status, dtype: int64
```

In [11]:
```
stroke_df['Residence_type'].value_counts()
```

Out[11]:
```
Urban    2596
Rural    2514
Name: Residence_type, dtype: int64
```

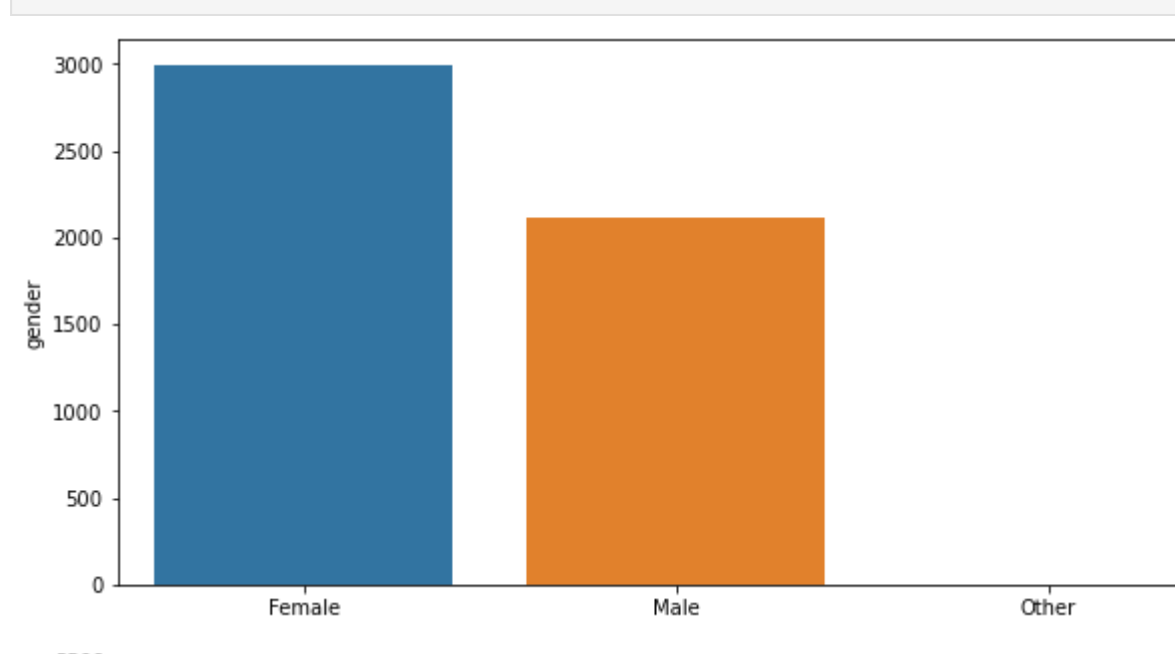
In [12]:
```
stroke_df.describe(include=['O'])# statistical statistics for categorical variables
```

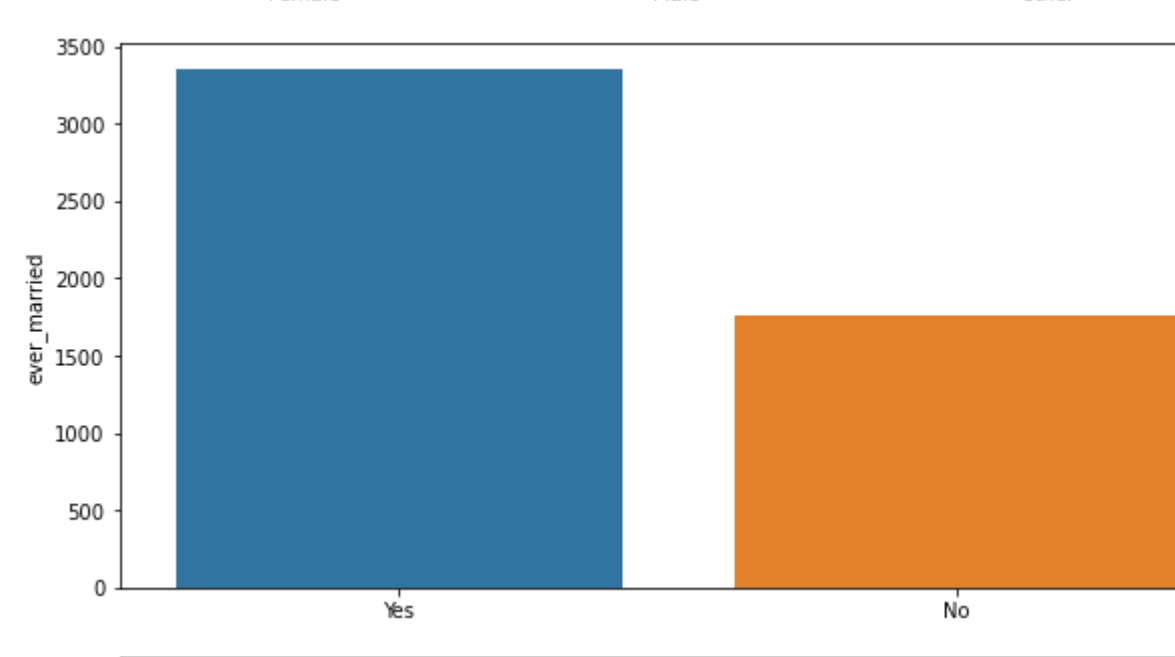
Out[12]:

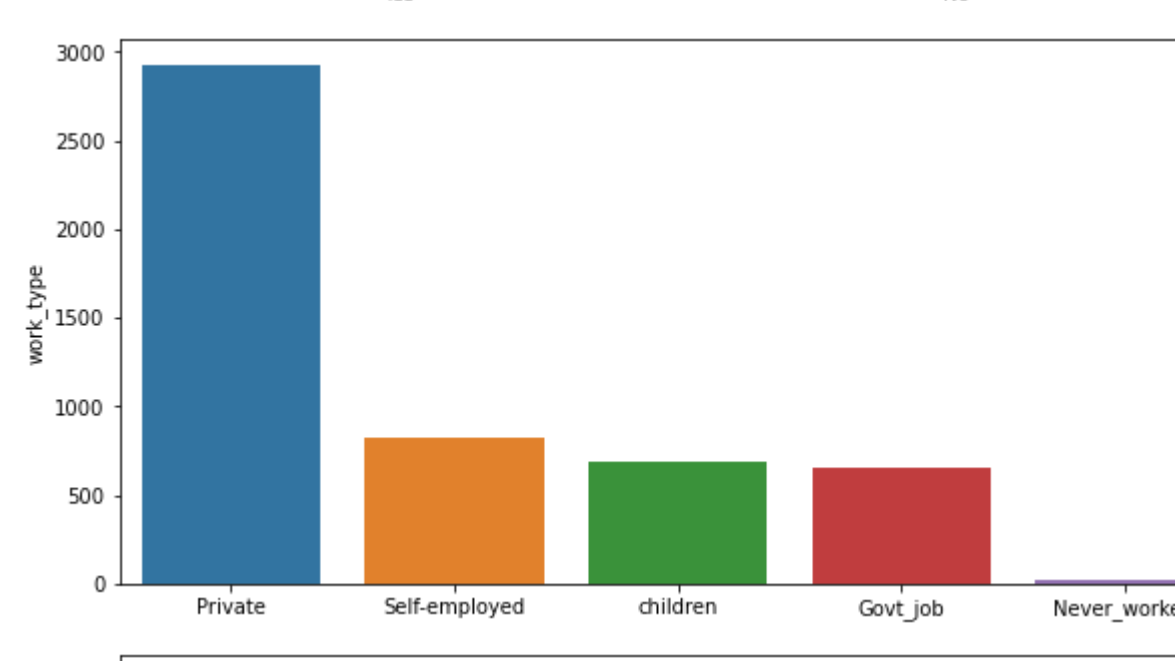
	gender	ever_married	work_type	Residence_type	smoking_status
count	5110	5110	5110	5110	5110
unique	3	2	5	2	4
top	Female	Yes	Private	Urban	never smoked
freq	2994	3353	2925	2596	1892

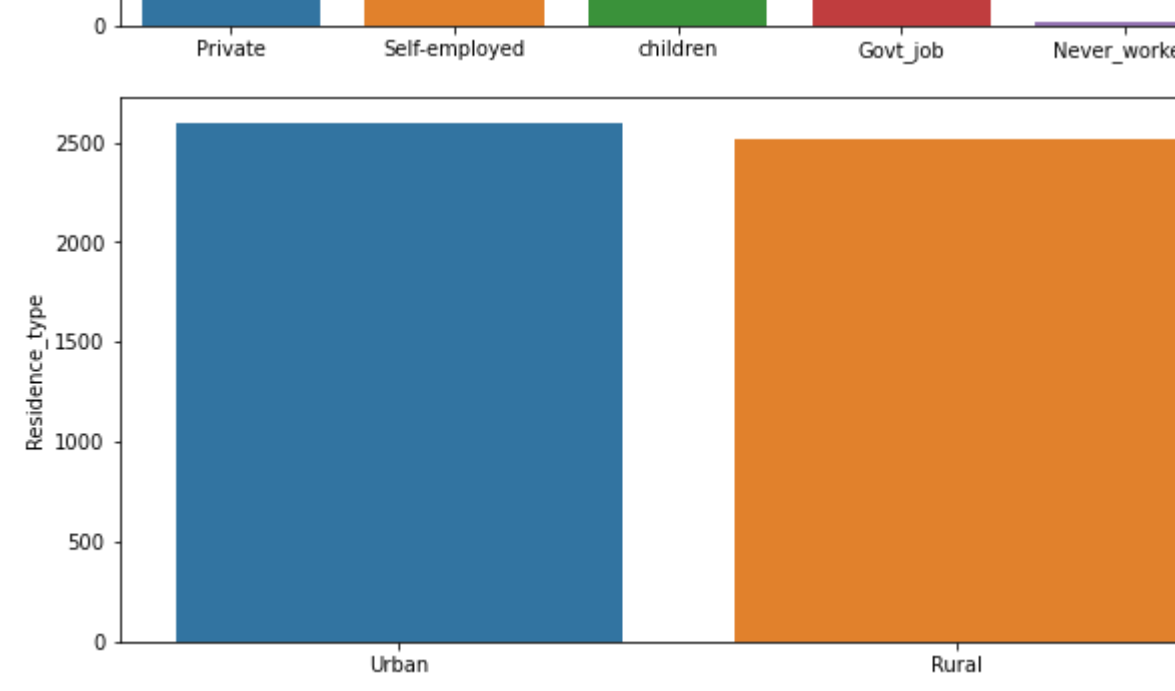
Bar plots can be used to compare distributions for categorical variables. We can visualize which subgroups tend to be the most common and how each group compares to one another.

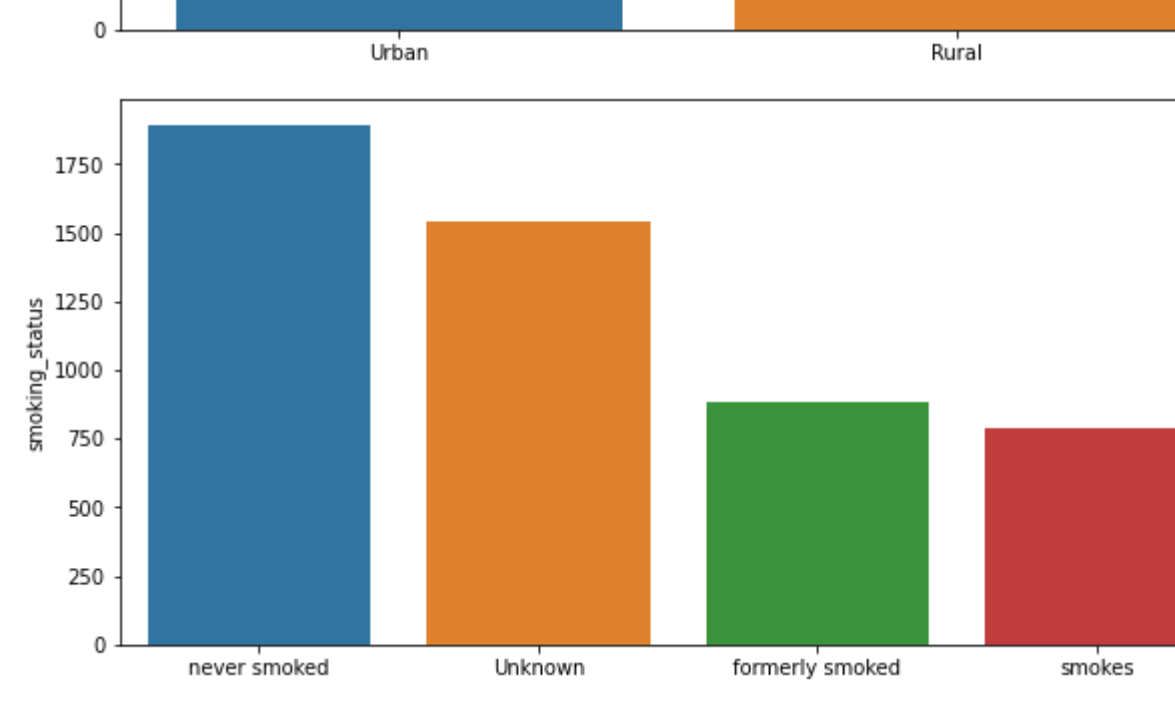
In [14]:
```
#plotting the categorical variables
stroke_barplot = stroke_df[['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status']]

for i in stroke_barplot.columns:
    plt.figure(figsize=(10,5))
    cat_num = stroke_df[i].value_counts()
    sns.barplot(x=cat_num.index, y=cat_num)
    plt.show()
```











In [15]:
```
stroke_df.isna().sum() # check for missing values bmi has 201 missing values 4% of the values are missing
```

Out[15]:
```
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
```

In [16]:
```
stroke_df[stroke_df.columns[stroke_df.isna().any()]] # further exploration on the missing bmi values
```

Out[16]:

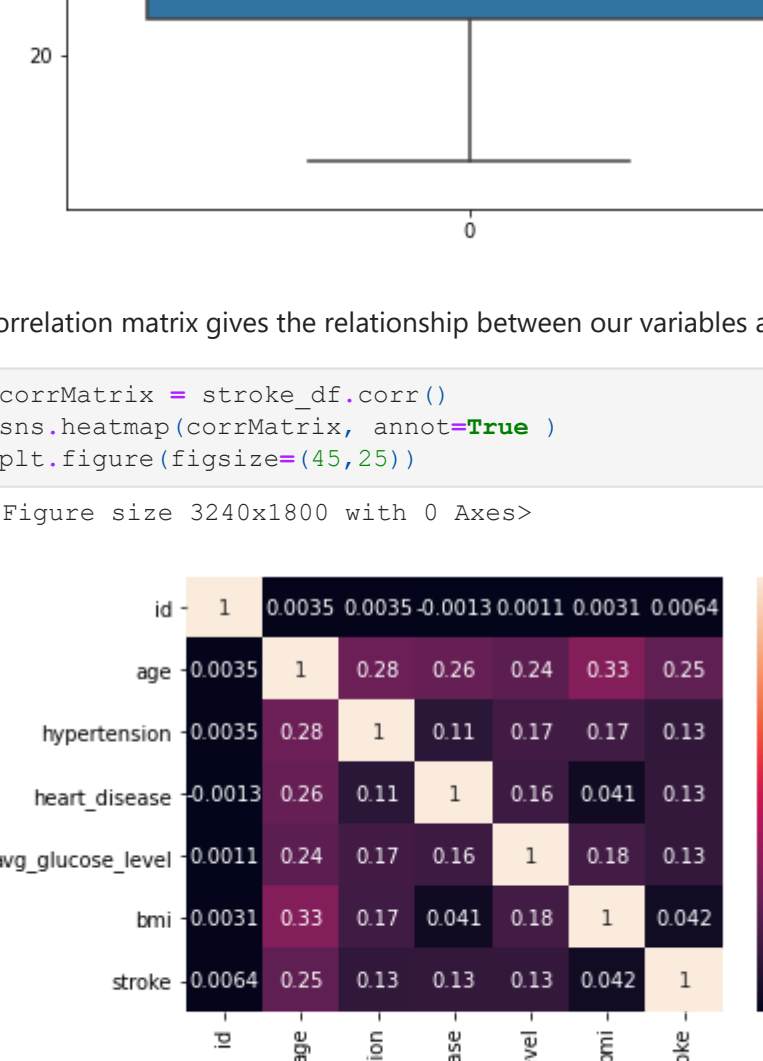
	bmi
0	36.6
1	NaN
2	32.5
3	34.4
4	24.0
5	29.0
6	27.4
7	22.8
8	NaN
9	24.2
10	29.7
11	36.8
12	27.3
13	NaN
14	28.2
15	30.9
16	37.5
17	25.8
18	37.8
19	NaN
20	22.4
21	48.9
22	26.6
23	32.5
24	27.2
25	23.5
26	28.2
27	NaN
28	28.3
29	NaN
...	...
5080	28.0
5081	40.8
5082	37.5
5083	24.2
5084	26.9
5085	33.1
5086	21.8
5087	34.7
5088	30.2
5089	16.8
5090	21.0
5091	30.9
5092	38.9
5093	NaN
5094	24.3
5095	17.4
5096	28.2
5097	40.8
5098	17.5
5099	NaN
5100	28.3
5101	24.5
5102	21.7
5103	46.9
5104	18.6
5105	NaN
5106	40.0
5107	30.6
5108	25.6
5109	26.2

5110 rows × 1 columns

A histogram can be used to visualize the distribution of numerical variables. We use a histogram to look at BMI in order to determine how missing values should be handled. As shown above, BMI has 201 missing values and must be handled appropriately.

In [17]:
```
stroke_df['bmi'].hist() #plot histogram for bmi
```

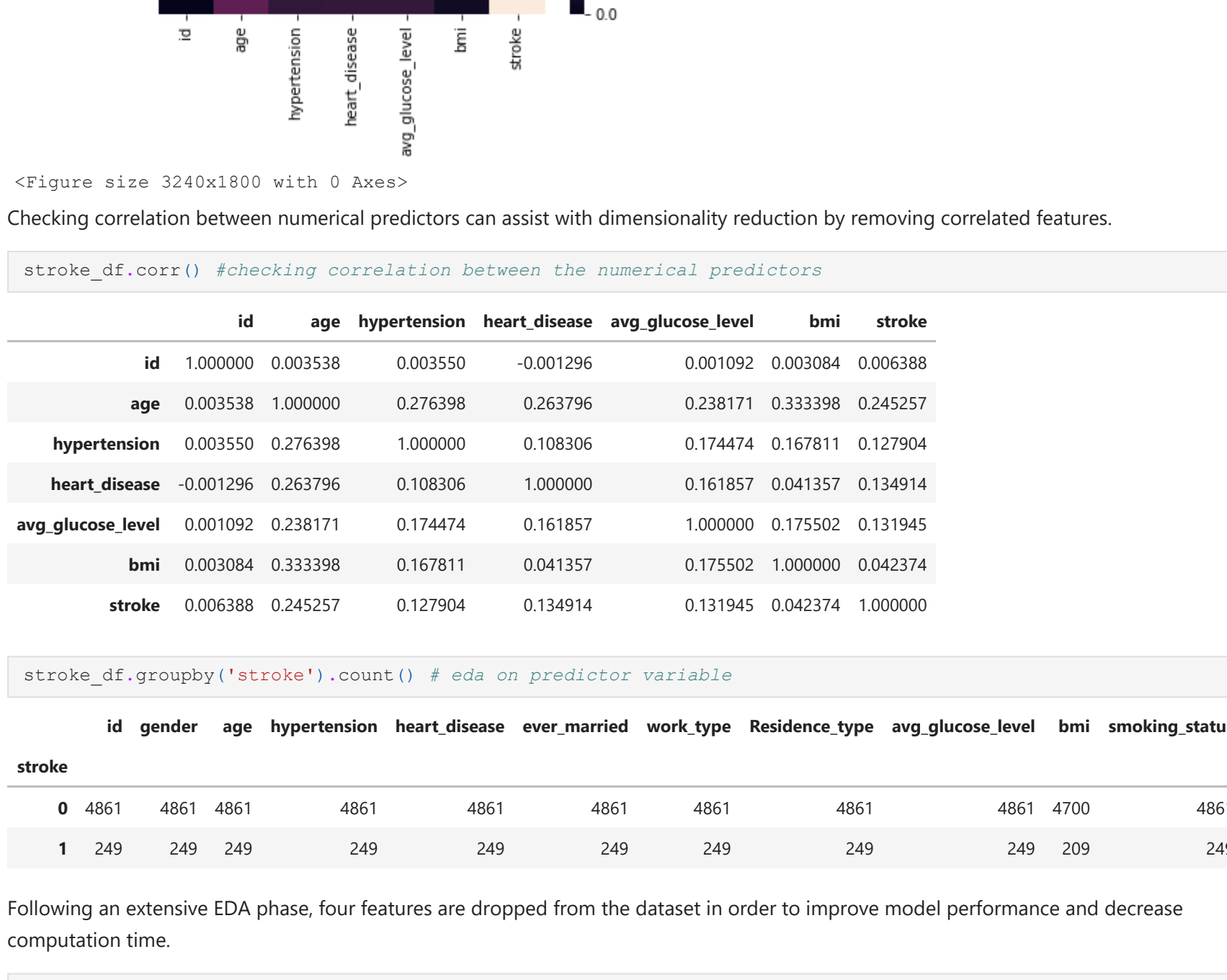
Out[17]:
```
<AxesSubplot:>
```



Further analyzing BMI, it appears that while median value for BMI is slightly lower for those who have not suffered from a stroke, there are more outliers and a larger range in values.

In [18]:
```
plt.figure(figsize=(15,10)) # boxplot for bmi
sns.boxplot(x='stroke',y='bmi',data=stroke_df)
```

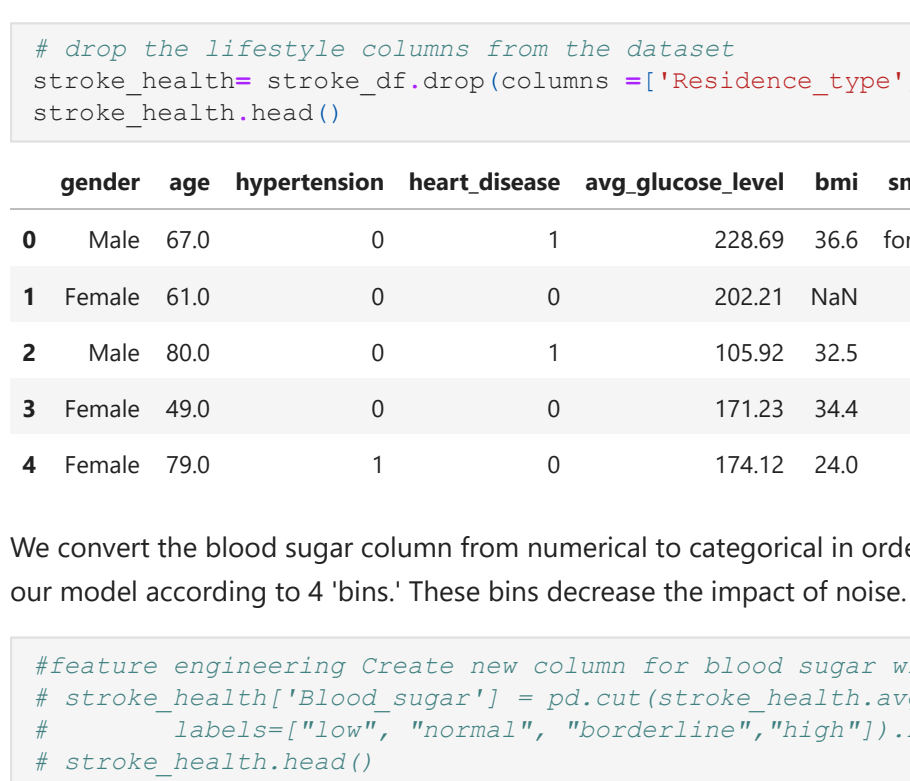
Out[18]:
```
<AxesSubplot:xlabel='stroke', ylabel='bmi'>
```



Correlation matrix gives the relationship between our variables and can assist with dimensionality reduction.

In [19]:
```
corrMatrix = stroke_df.corr()
sns.heatmap(corrMatrix, annot=True )
plt.figure(figsize=(45,25))
```

Out[19]:
```
<Figure size 3240x1800 with 0 Axes>
```



```
<Figure size 3240x1800 with 0 Axes>
```

Checking correlation between numerical predictors can assist with dimensionality reduction by removing correlated features.

In [20]:
```
stroke_df.corr() #checking correlation between the numerical predictors
```

Out[20]:

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
id	1.000000	0.003538	0.003550	-0.001296	0.001092	0.003084	0.006388
age	0.003538	1.000000	0.276398	0.263796	0.238171	0.333398	0.245257
hypertension	0.003550	0.276398	1.000000	0.108306	0.174474	0.167811	0.127904
heart_disease	-0.001296	0.263796	0.108306	1.000000	0.161857	0.041357	0.134914
avg_glucose_level	0.001092	0.238171	0.174474	0.161857	1.000000	0.175502	0.131945
bmi	0.003084	0.333398	0.167811	0.041357	0.175502	1.000000	0.042374
stroke	0.006388	0.245257	0.127904	0.134914	0.131945	0.042374	1.000000

In [21]:
```
stroke_df.groupby('stroke').count() # eda on predictor variable
```

Out[21]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
stroke											
0	4861	4861	4861	4861	4861	4861	4861	4861	4861	4700	4861
1	249	249	249	249	249	249	249	249	249	209	249

Following an extensive EDA phase, four features are dropped from the dataset in order to improve model performance and decrease computation time.

In [26]:
```
# drop the lifestyle columns from the dataset
stroke_health= stroke_df.drop(columns =['Residence_type', 'work_type', 'ever_married','id'])
stroke_health.head()
```

Out[26]:

	gender	age	hypertension	heart_disease	avg_glucose_level	bmi	smoking_status	stroke
0	Male	67.0	0	1	228.69	36.6	formerly smoked	1
1	Female	61.0	0	0	202.21	NaN	never smoked	1
2	Male	80.0	0	1	105.92	32.5	never smoked	1
3	Female	49.0	0	0	171.23	34.4	smokes	1
4	Female	79.0	1	0	174.12	24.0	never smoked	1

We convert the blood sugar column from numerical to categorical in order to improve signal to noise ratio. As a result, we are able to fit our model according to 4 'bins.' These bins decrease the impact of noise. The four bins are: low, normal, borderline and high.

In [27]:
```
#feature engineering Create new column for blood sugar with 4 categories
# stroke_health['Blood_sugar'] = pd.cut(stroke_health.avg_glucose_level, bins=4,
#        labels=["low", "normal", "borderline","high"]).head()
# stroke_health.head()

stroke_health['Blood_sugar'] = pd.cut(stroke_health.avg_glucose_level, bins=[0,50,72,150,271],
       labels=["low", "normal", "borderline","high"]).head()
stroke_health.head()
```

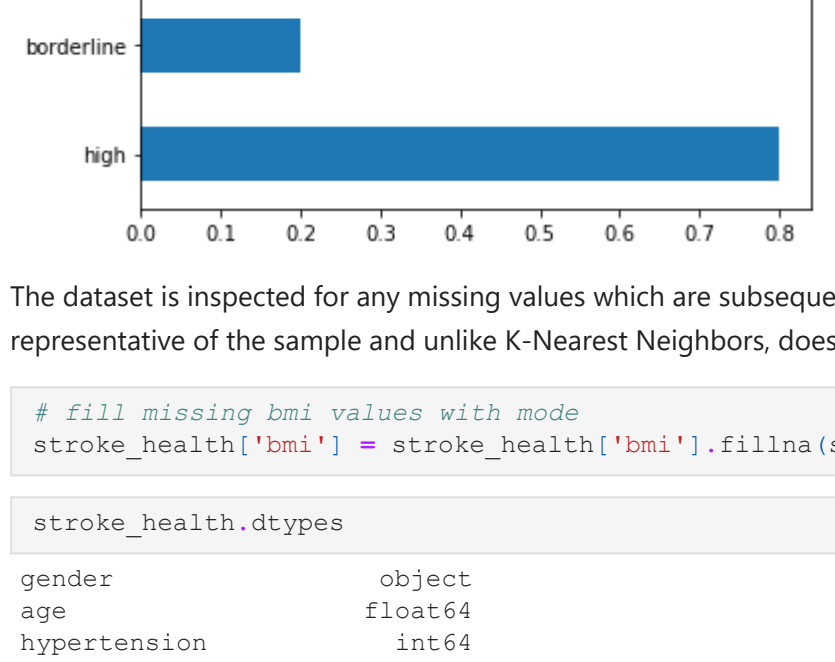
Out[27]:

	gender	age	hypertension	heart_disease	avg_glucose_level	bmi	smoking_status	stroke	Blood_sugar
0	Male	67.0	0	1	228.69	36.6	formerly smoked	1	high
1	Female	61.0	0	0	202.21	NaN	never smoked	1	high
2	Male	80.0	0	1	105.92	32.5	never smoked	1	borderline
3	Female	49.0	0	0	171.23	34.4	smokes	1	high
4	Female	79.0	1	0	174.12	24.0	never smoked	1	high

In [28]:
```
stroke_health['Blood_sugar'].value_counts(normalize=True)
```

Out[28]:
```
high          0.8
borderline    0.2
normal        0.0
low           0.0
Name: Blood_sugar, dtype: float64
```

In [29]:
```
#plot blood sugar
stroke_health['Blood_sugar'].value_counts(normalize=True).plot(kind='barh')
plt.show()
```



The dataset is inspected for any missing values which are subsequently imputed by the median value. This approach is the most representative of the sample and unlike K-Nearest Neighbors, does not yield negative values.

In [30]:
```
# fill missing bmi values with mode
stroke_health['bmi'] = stroke_health['bmi'].fillna(stroke_health['bmi'].mode()[0])
```

In [31]:
```
stroke_health.dtypes
```

Out[31]:
```
gender                 object
age                   float64
hypertension            int64
heart_disease           int64
avg_glucose_level     float64
bmi                   float64
smoking_status         object
stroke                  int64
Blood_sugar          category
dtype: object
```