In [1]:

```
import numpy as np
import pandas as pd
import os
for dirname, _, filenames in os.walk('D:\stroke\healthcare-dataset-stroke-data.csv'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [2]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```
df = pd.read_csv('D:\stroke\healthcare-dataset-stroke-data.csv')
df.head(10)
```

Out[3]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_ty
0	9046	Male	67.0	0	1	Yes	Private	Urb
1	51676	Female	61.0	0	0	Yes	Self-employed	Ru
2	31112	Male	80.0	0	1	Yes	Private	Ru
3	60182	Female	49.0	0	0	Yes	Private	Urb
4	1665	Female	79.0	1	0	Yes	Self-employed	Ru
5	56669	Male	81.0	0	0	Yes	Private	Urb
6	53882	Male	74.0	1	1	Yes	Private	Ru
7	10434	Female	69.0	0	0	No	Private	Urb
8	27419	Female	59.0	0	0	Yes	Private	Ru
9	60491	Female	78.0	0	0	Yes	Private	Urb

In [4]:

```
print(df.columns.tolist())
```

```
['id', 'gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
'work_type', 'Residence_type', 'avg_glucose_level', 'bmi', 'smoking_statu
s', 'stroke']
```

In [5]:

```
df.shape
# find how many data we have and how many type of data
# 5110 rows = total data , 12 columns = 12 types
```

Out[5]:

```
(5110, 12)
```

In [6]:

```
df.info()
# check whether there are null value
```

```
<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 [7]:

```
data = df.drop('id', axis=1) # id is useless, so we drop it
data.isnull().sum() # find how many null value in bmi
```

Out[7]:

```
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 [8]:

```
data['bmi'].fillna(data['bmi'].mean(), inplace=True)
# replace null value by mean
```

In [9]:

```
data.isnull().sum()
```

Out[9]:

```
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
```

In [10]:

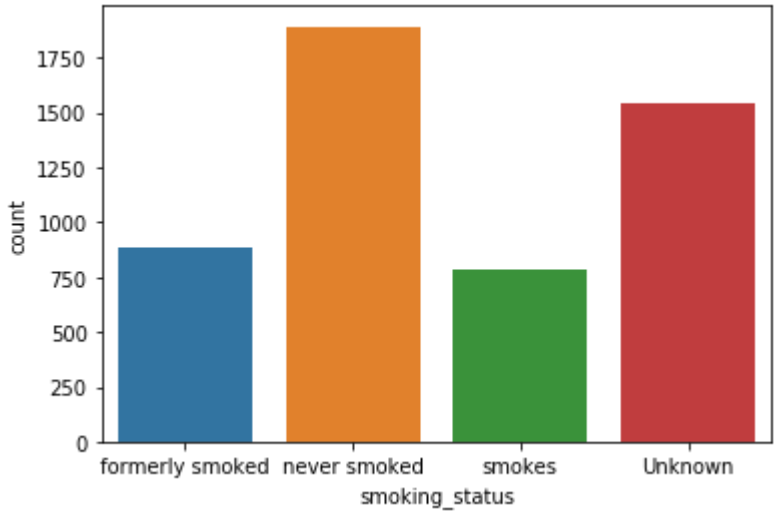
```
features = ['gender', 'hypertension', 'heart_disease','ever_married', 'work_type', 'Res
idence_type', 'smoking_status']
for feature in features:
    print(data[feature].unique())
# find unique value
# like in gender: Other(only 1, so ignore)
# like in smoking_status: unknown(looks like there are a lot, I used mode to replace th
em)
```

```
['Male' 'Female' 'Other']
[0 1]
[1 0]
['Yes' 'No']
['Private' 'Self-employed' 'Govt_job' 'children' 'Never_worked']
['Urban' 'Rural']
['formerly smoked' 'never smoked' 'smokes' 'Unknown']
```

In [11]:

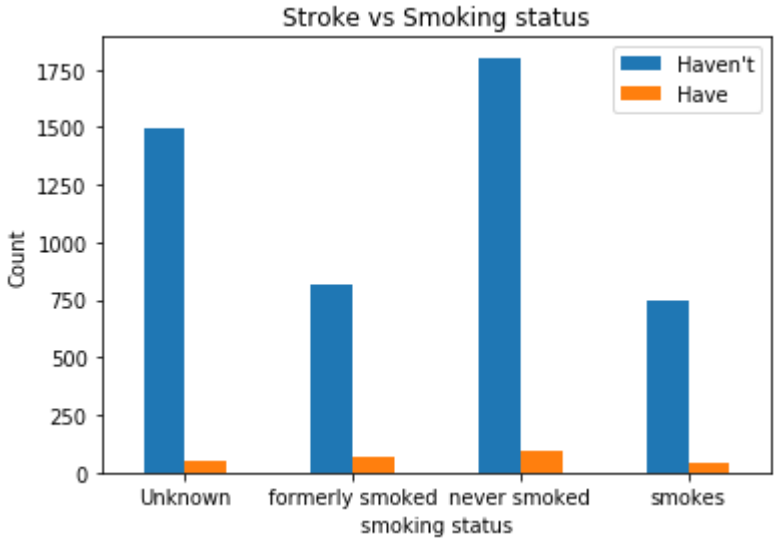
```
print(data.smoking_status.value_counts())
sns.countplot(x = data['smoking_status'])
plt.show()
```

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



In [12]:

```
pd.crosstab(data.smoking_status,data.stroke).plot(kind="bar")
plt.title('Stroke vs Smoking status')
plt.xlabel('smoking status')
plt.xticks(rotation = 0)
plt.legend(["Haven't", "Have"])
plt.ylabel('Count')
plt.show()
```



In [13]:

```
data['smoking_status'].replace('Unknown', np.nan, inplace=True)
data['smoking_status'].fillna(data['smoking_status'].mode()[0], inplace = True)
```

In [14]:

```
data.describe()
```

Out[14]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	strok
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	5110.00000
mean	43.226614	0.097456	0.054012	106.147677	28.893237	0.04872
std	22.612647	0.296607	0.226063	45.283560	7.698018	0.21532
min	0.080000	0.000000	0.000000	55.120000	10.300000	0.00000
25%	25.000000	0.000000	0.000000	77.245000	23.800000	0.00000
50%	45.000000	0.000000	0.000000	91.885000	28.400000	0.00000
75%	61.000000	0.000000	0.000000	114.090000	32.800000	0.00000
max	82.000000	1.000000	1.000000	271.740000	97.600000	1.00000

In [15]:

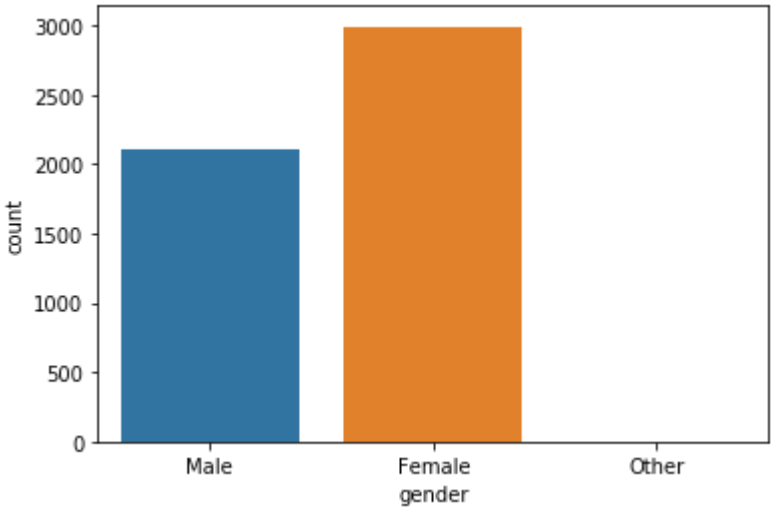
```
data.corr()
# varibales by varibles
```

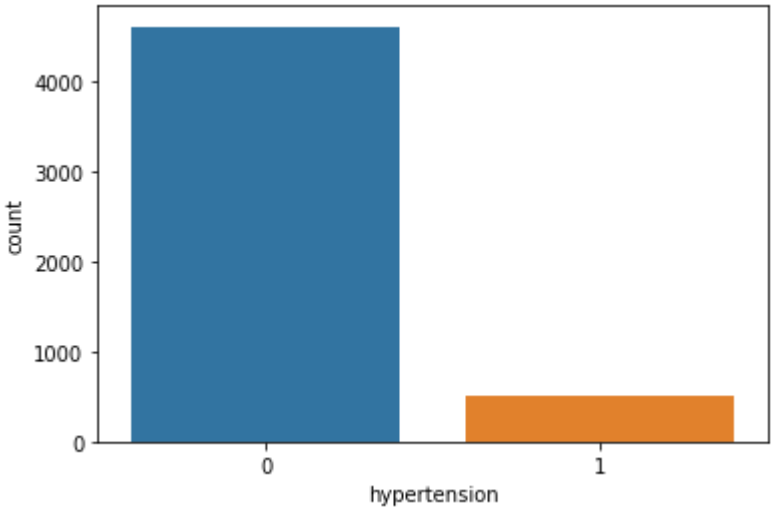
Out[15]:

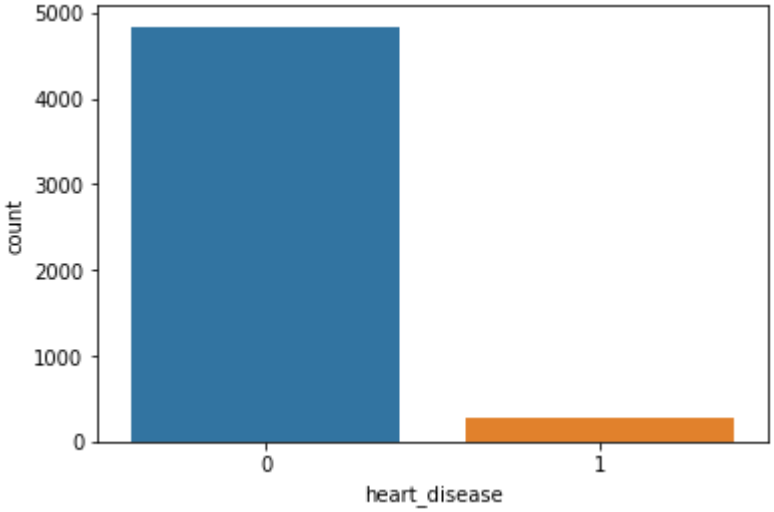
	age	hypertension	heart_disease	avg_glucose_level	bmi	str
age	1.000000	0.276398	0.263796	0.238171	0.325942	0.245
hypertension	0.276398	1.000000	0.108306	0.174474	0.160189	0.127
heart_disease	0.263796	0.108306	1.000000	0.161857	0.038899	0.134
avg_glucose_level	0.238171	0.174474	0.161857	1.000000	0.168751	0.131
bmi	0.325942	0.160189	0.038899	0.168751	1.000000	0.038
stroke	0.245257	0.127904	0.134914	0.131945	0.038947	1.000

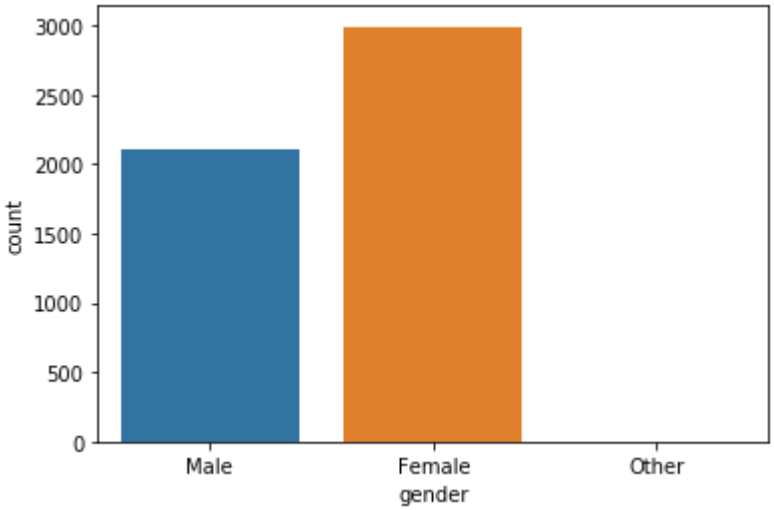
In [16]:

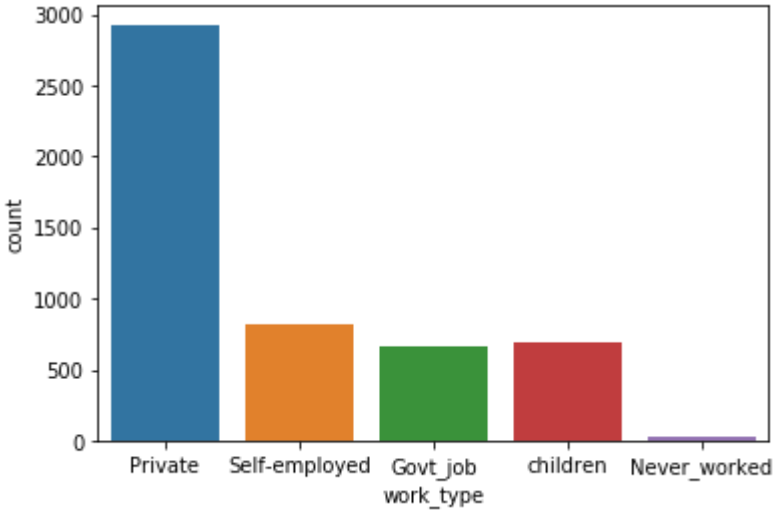
```
for feature in features:
    plt.figure()
    sns.countplot(x = data[feature])
    plt.show()
```

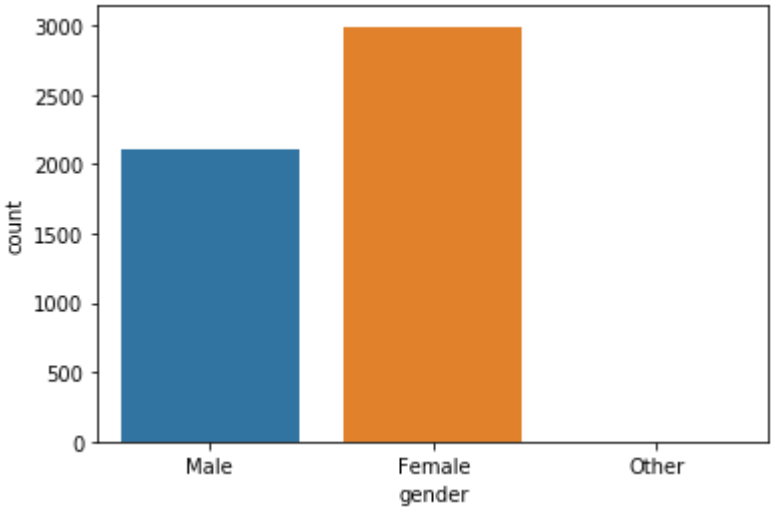

3000
2500
2000
1500
1000
500
0
count
Male
Female
Other
gender

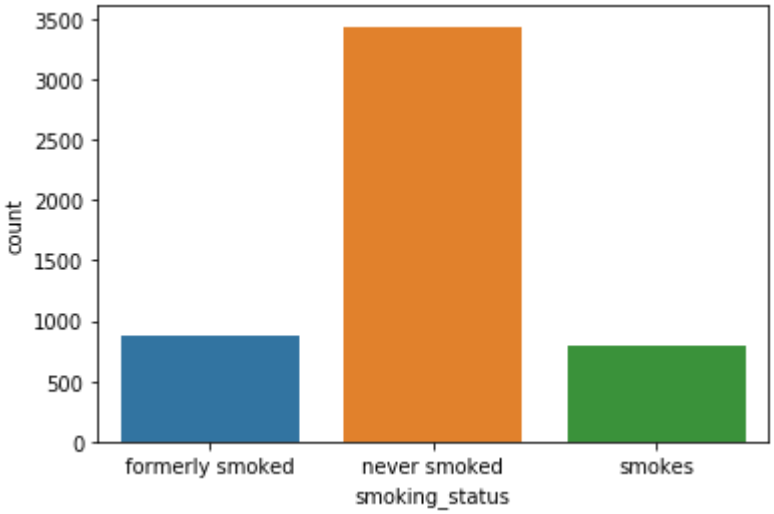

4000
3000
2000
1000
0
count
0
1
hypertension


5000
4000
3000
2000
1000
0
count
0
1
heart_disease


3000
2500
2000
1500
1000
500
0
count
Male
Female
Other
gender


3000
2500
2000
1500
1000
500
0
count
Private
Self-employed
Govt_job
children
Never_worked
work_type


3000
2500
2000
1500
1000
500
0
count
Male
Female
Other
gender


3500
3000
2500
2000
1500
1000
500
0
count
formerly smoked
never smoked
smokes
smoking_status

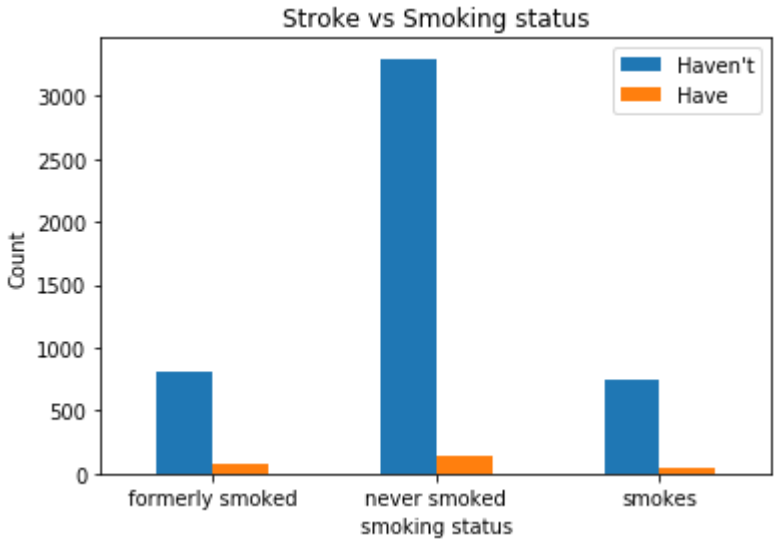
In [17]:

```
print(data.hypertension.value_counts())
print("------------------------")
print(data.heart_disease.value_counts())
print("------------------------")
print(data.ever_married.value_counts())
print("------------------------")
print(data.gender.value_counts())
print("------------------------")
print(data.work_type.value_counts())
print("------------------------")
print(data.Residence_type.value_counts())
print("------------------------")
print(data.smoking_status.value_counts())
print("------------------------")
print(data.stroke.value_counts())
```

```
0    4612
1     498
Name: hypertension, dtype: int64
------------------
0    4834
1     276
Name: heart_disease, dtype: int64
------------------
Yes    3353
No     1757
Name: ever_married, dtype: int64
--------------------
Female    2994
Male      2115
Other        1
Name: gender, dtype: int64
--------------------
Private          2925
Self-employed     819
children          687
Govt_job          657
Never_worked       22
Name: work_type, dtype: int64
------------------------
Urban    2596
Rural    2514
Name: Residence_type, dtype: int64
------------------
never smoked       3436
formerly smoked     885
smokes              789
Name: smoking_status, dtype: int64
------------------
0    4861
1     249
Name: stroke, dtype: int64
```

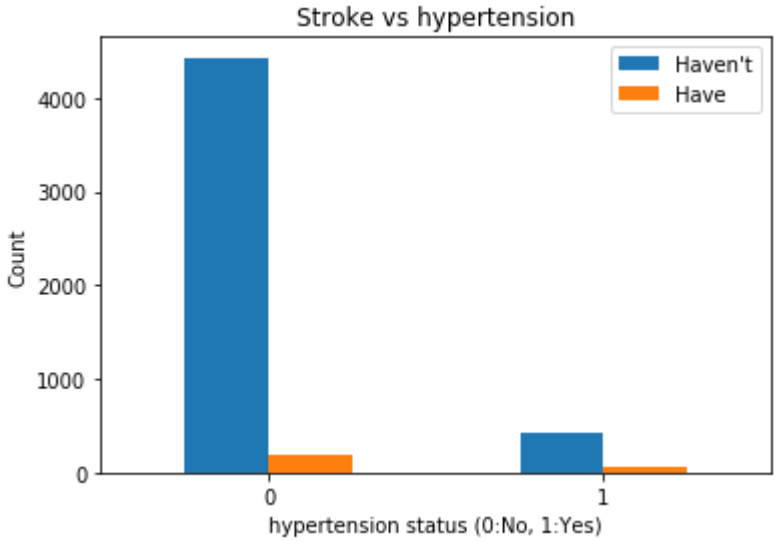
In [18]:

```
pd.crosstab(data.smoking_status,data.stroke).plot(kind="bar")
plt.title('Stroke vs Smoking status')
plt.xlabel('smoking status')
plt.xticks(rotation = 0)
plt.legend(["Haven't", "Have"])
plt.ylabel('Count')
plt.show()
#we can see smoking status is not relevant to stroke
```



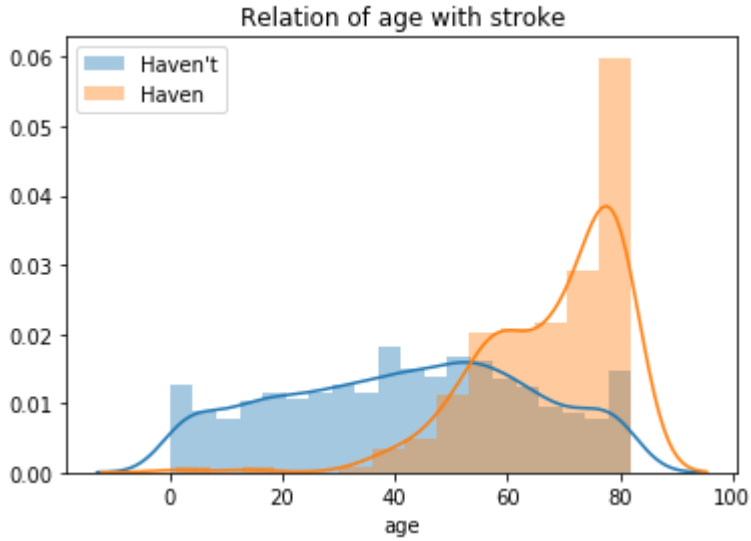
In [19]:

```
pd.crosstab(data.hypertension,data.stroke).plot(kind="bar")
plt.title('Stroke vs hypertension')
plt.xlabel('hypertension status (0:No, 1:Yes)')
plt.xticks(rotation = 0)
plt.legend(["Haven't", "Have"])
plt.ylabel('Count')
plt.show()
#we can see shypertension is not relevant to stroke
```



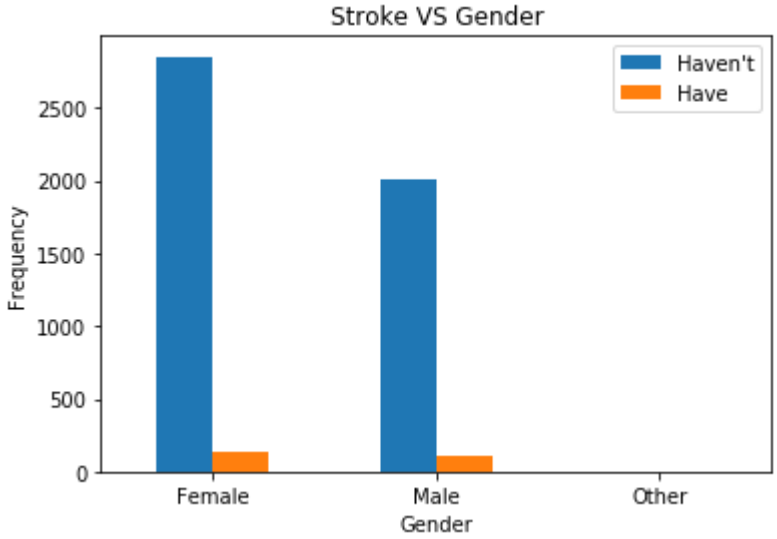
In [20]:

```
plt.title('Relation of age with stroke')
sns.distplot(data.age[data.stroke==0])
sns.distplot(data.age[data.stroke==1])
plt.legend(["Haven't","Haven"])
plt.show()
#looks like elderly will easily get stroke
```



In [21]:

```
pd.crosstab(data.gender,data.stroke).plot(kind="bar")
plt.title('Stroke VS Gender')
plt.xlabel('Gender')
plt.xticks(rotation = 0)
plt.legend(["Haven't", "Have"])
plt.ylabel('Frequency')
plt.show()
```

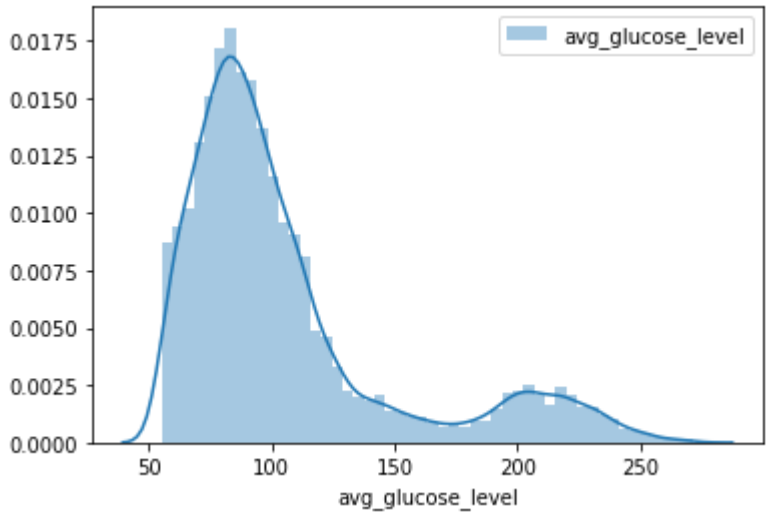


In [22]:

```
plt.figure()
sns.distplot(data["avg_glucose_level"], label="avg_glucose_level")
plt.legend()
```

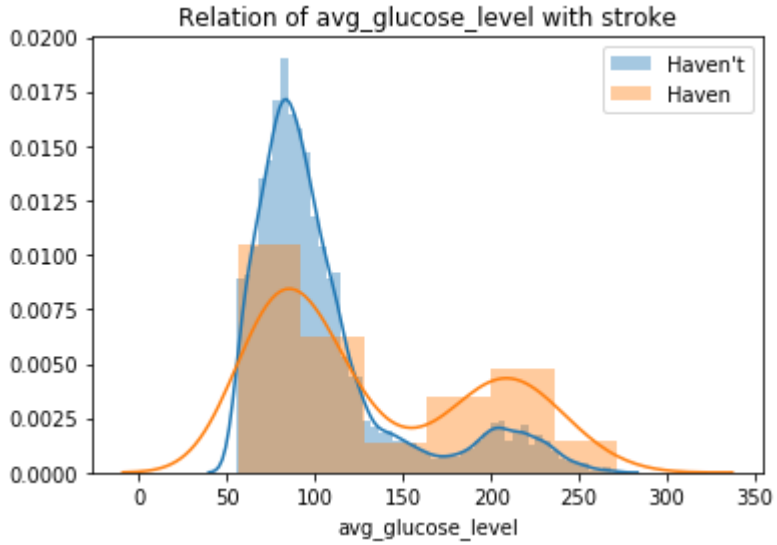
Out[22]:

```
<matplotlib.legend.Legend at 0x208676f9448>
```



In [23]:

```
plt.title('Relation of avg_glucose_level with stroke')
sns.distplot(data.avg_glucose_level[data.stroke==0])
sns.distplot(data.avg_glucose_level[data.stroke==1])
plt.legend(["Haven't","Haven"])
plt.show()
```

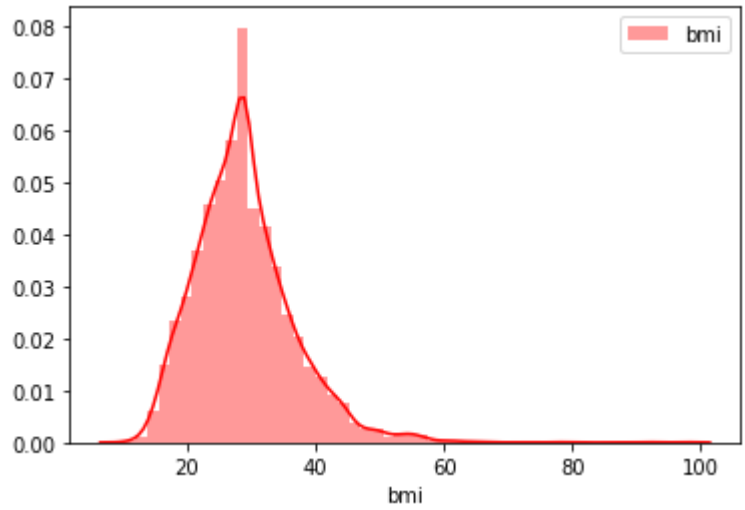


In [24]:

```
plt.figure()
sns.distplot(data["bmi"], label="bmi",color="red")
plt.legend()
```

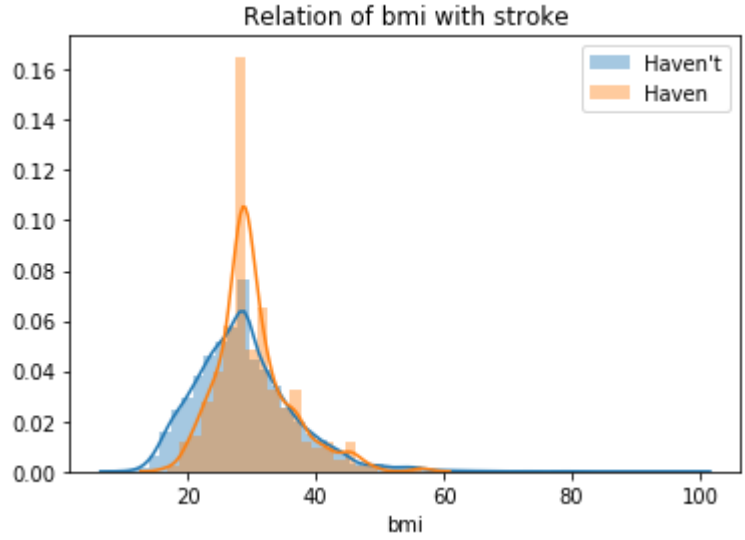
Out[24]:

```
<matplotlib.legend.Legend at 0x208678dabc8>
```



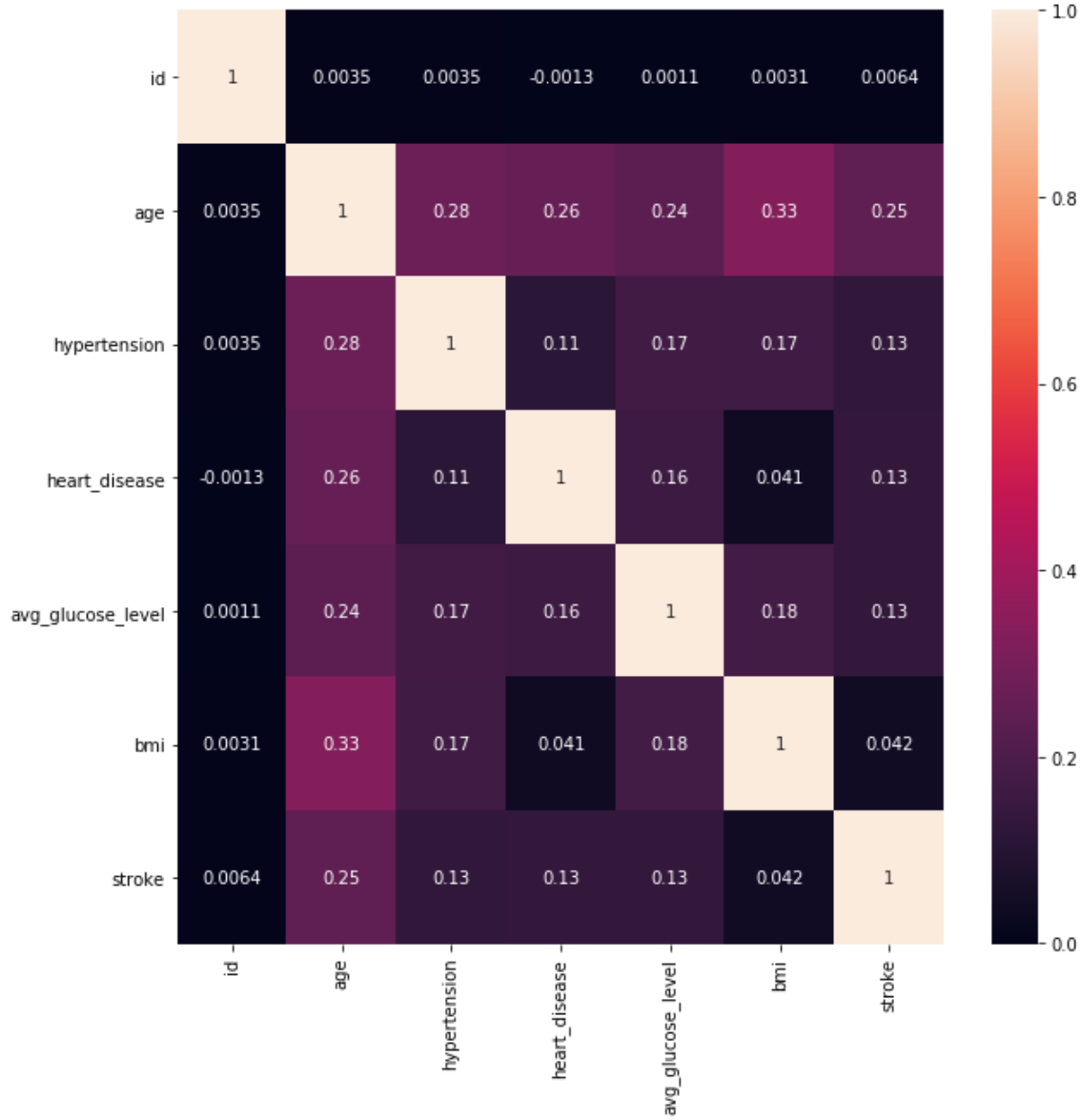
In [25]:

```
plt.title('Relation of bmi with stroke')
sns.distplot(data.bmi[data.stroke==0])
sns.distplot(data.bmi[data.stroke==1])
plt.legend(["Haven't","Haven"])
plt.show()
```



In [26]:

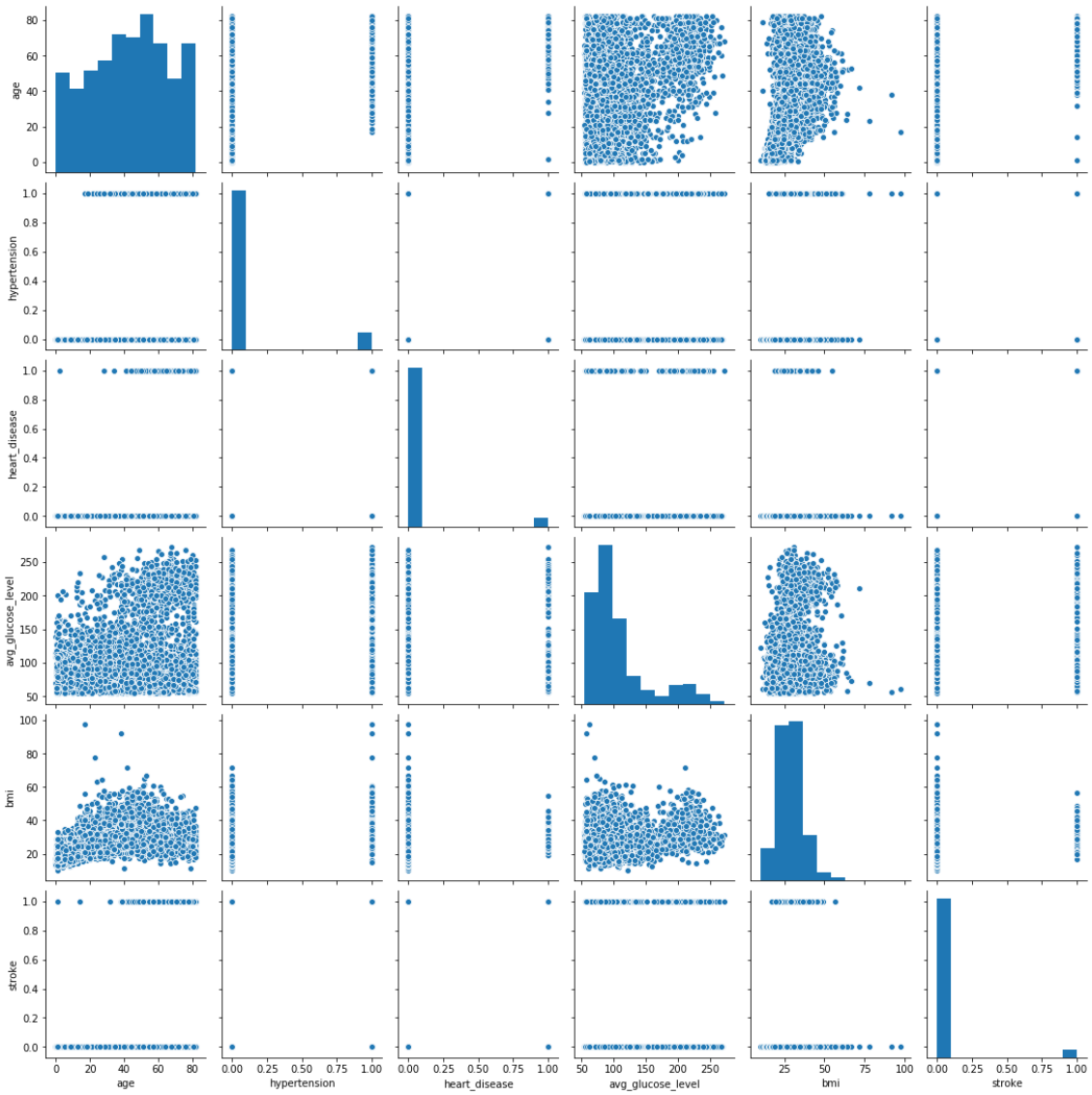
```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(),annot=True);
```



In [27]:

```
plt.figure(figsize=(10,10))
sns.pairplot(data)
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



In [28]:

```
from sklearn.preprocessing import LabelEncoder
lebalencode = LabelEncoder()
en_data = data.apply(lebalencode.fit_transform)
en_data.head()
# label encoding
```

Out[28]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_
0	1	88	0	1	1	2	1	
1	0	82	0	0	1	3	0	
2	1	101	0	1	1	2	0	
3	0	70	0	0	1	2	1	
4	0	100	1	0	1	3	0	

In [29]:

```
x = en_data.drop('stroke', axis = 1)
y = en_data['stroke']
print('X Shape: ', x.shape)
print('Y Shape: ', y.shape)
# feature selection and see now how many rows and columns
# after drop stroke, only 10 types data
```

```
X Shape:  (5110, 10)
Y Shape:  (5110,)
```

In [30]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size= 0.3, random_state=
0)
# split data into 30% and 70&
```

In [31]:

```
print("Transactions: ")
print("x_train: ", x_train.shape)
print("y_train: ", y_train.shape)
print("x_test: ", x_test.shape)
print("y_test: ", y_test.shape)
# To see the shape of test dataset and train dataset
```

```
Transactions:
x_train:  (3577, 10)
y_train:  (3577,)
x_test:  (1533, 10)
y_test:  (1533,)
```

In [32]:

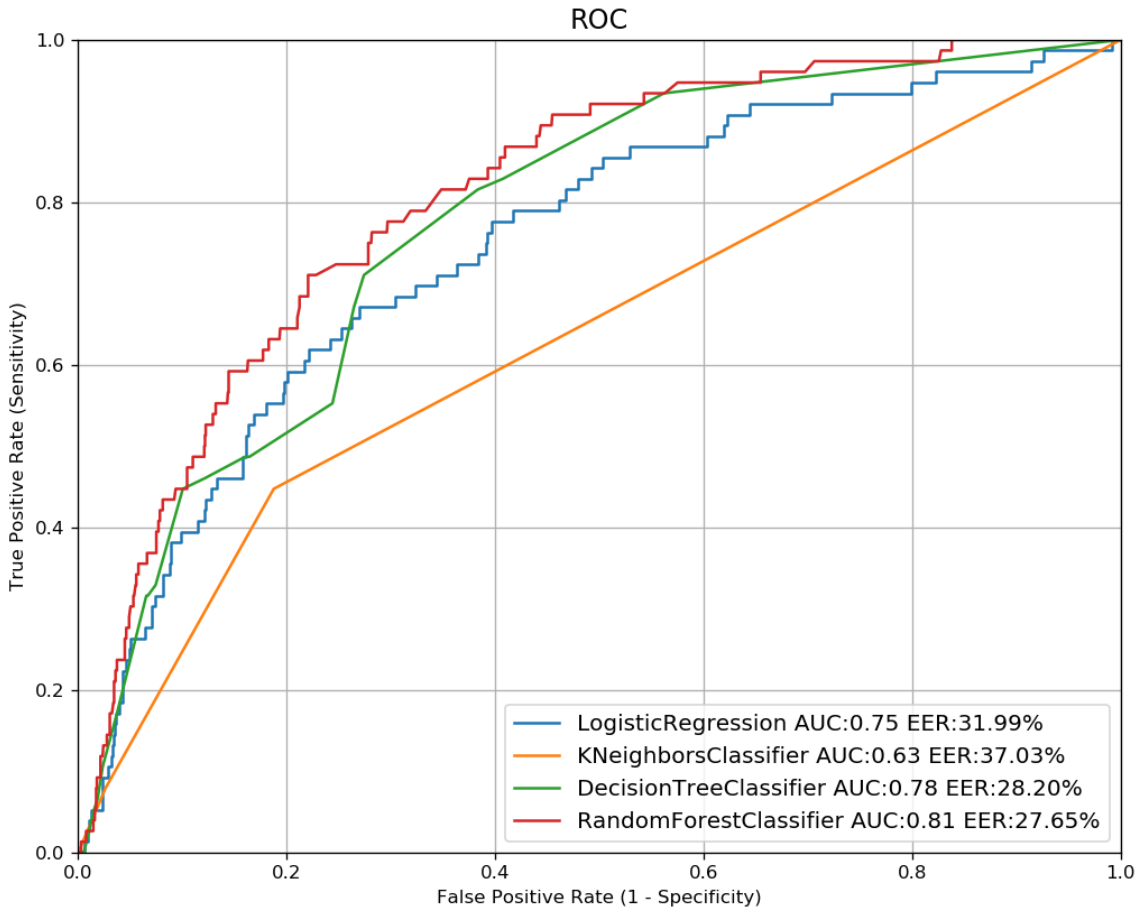
```
from sklearn.ensemble import RandomForestClassifier #for the model
from sklearn.tree import DecisionTreeClassifier #for the model
from sklearn.neighbors import KNeighborsClassifier #for the model
from sklearn.linear_model import LogisticRegression #for the model

from sklearn.metrics import roc_curve, auc #for model evaluation
from sklearn.metrics import classification_report #for model evaluation
from sklearn.metrics import confusion_matrix #for model evaluation
from sklearn.model_selection import train_test_split #for data splitting
```

In [33]:

```
# Evaluation with ROC curve
def plot_roc(y_test,y_proba, model_name):
    fpr, tpr, thresholds = roc_curve(y_test, y_proba)
    fnr = 1 - tpr
    eer_idx = np.nanargmin(np.absolute((fnr - fpr)))
    eer = (fpr[eer_idx] + fnr[eer_idx]) / 2
    area = auc(fpr, tpr)
    label="{} AUC:{:.2f} EER:{:.2f}%".format(model_name,area,eer * 100)
    plt.plot(fpr, tpr, label=label)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.0])
    plt.rcParams['font.size'] = 12
    plt.title('ROC')
    plt.xlabel('False Positive Rate (1 - Specificity)')
    plt.ylabel('True Positive Rate (Sensitivity)')
    plt.legend()
    plt.grid(True)
```

In [34]:

```
plt.figure(figsize=(10,8),dpi=120)
classifiers = [
    LogisticRegression(C=0.1,penalty='l2',random_state=0),
    KNeighborsClassifier(5),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=10)]
for model in classifiers:
    model.fit(x_train,y_train)
    y_proba=model.predict_proba(x_test)[:,1]
    plot_roc(y_test,y_proba,type(model).__name__)
```



In [35]:

```
from sklearn.preprocessing import StandardScaler
sscaler = StandardScaler()
x_train = sscaler.fit_transform(x_train)
x_test = sscaler.transform(x_test)
# Standardization
```

In [36]:

```
# use randomforest as our model
randomforest = RandomForestClassifier()
randomforest.fit(x_train, y_train)
y_pred = randomforest.predict(x_test)
```

In [37]:

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
print("The Classification report: \n{}".format(classification_report(y_test, y_pred)))
print("---------------------------------------------------------------------------------------------------------- ")
print("The Accuracy Score is: {:.3f}%".format(accuracy_score(y_test, y_pred)*100))
```

```
The Classification report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97      1457
           1       0.00      0.00      0.00        76

    accuracy                           0.95      1533
   macro avg       0.48      0.50      0.49      1533
weighted avg       0.90      0.95      0.93      1533

----------------------------------------------------------------------
The Accuracy Score is: 94.977%
```

In [38]:

```
importances = randomforest.feature_importances_
feat_labels = x.columns[0:]
#  find the feature importances
```

In [39]:

```
indices = np.argsort(importances)
```

In [40]:

```
for f in range(x_train.shape[1]):
    print("%2d) %-*s %f" % (f + 1, 30, feat_labels[indices[f]], importances[indices[f
]]))
    # show each featuers' importrances to stroke
```

```
 1) ever_married                   0.020735
 2) hypertension                   0.024168
 3) heart_disease                  0.029258
 4) Residence_type                 0.031888
 5) gender                         0.034417
 6) smoking_status                 0.051633
 7) work_type                      0.053477
 8) bmi                            0.234667
 9) age                            0.240199
10) avg_glucose_level              0.279557
```

In [41]:

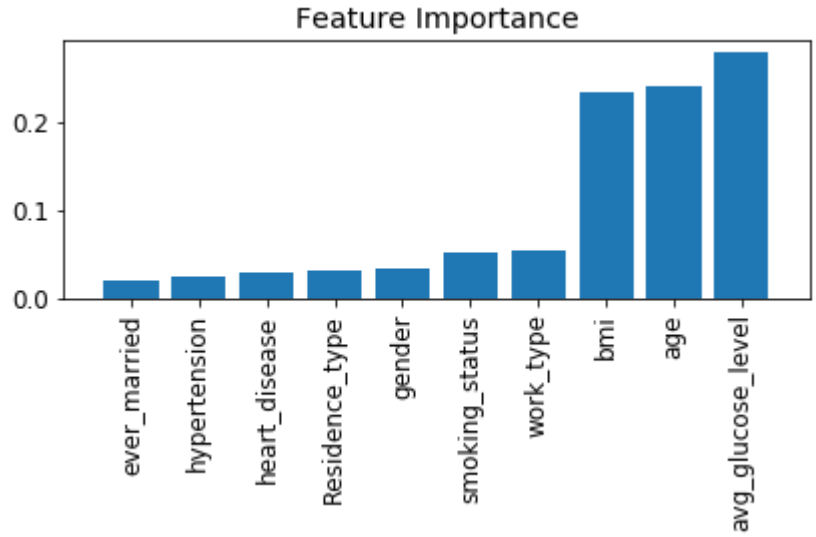
```
threshold = 0.15
x_selected = x_train[:, importances > threshold]
x_selected.shape
# age, bmi and avg_glucose_level are most significant value
```

Out[41]:

```
(3577, 3)
```

In [42]:

```
plt.title('Feature Importance')
plt.bar(range(x_train.shape[1]),
        importances[indices],
        align='center')
plt.xticks(range(x_train.shape[1]),
          feat_labels[indices], rotation=90)
plt.xlim([-1,x_train.shape[1]])
plt.tight_layout()
plt.show()
```



In [ ]: