In [9]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [10]:

```
df = pd.read_csv("healthcare-dataset-stroke-data.csv")
df
```

Out[10]:

	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
...	...	...	...	...	...	...	...	
5105	18234	Female	80.0	1	0	Yes	Private	Urb
5106	44873	Female	81.0	0	0	Yes	Self-employed	Urb
5107	19723	Female	35.0	0	0	Yes	Self-employed	Ru
5108	37544	Male	51.0	0	0	Yes	Private	Ru
5109	44679	Female	44.0	0	0	Yes	Govt_job	Urb

5110 rows × 12 columns

In [11]:

```
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
```

Attribute Information

1) id: unique identifier
2) gender: "Male", "Female" or "Other"
3) age: age of the patient
4) hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
5) heart_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
6) ever_married: "No" or "Yes"
7) work_type: "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
8) Residence_type: "Rural" or "Urban"
9) avg_glucose_level: average glucose level in blood
10) bmi: body mass index
11) smoking_status: "formerly smoked", "never smoked", "smokes" or "Unknown"*
12) stroke: 1 if the patient had a stroke or 0 if not
*Note: "Unknown" in smoking_status means that the information is unavailable for this patient

In [12]:

```
df.describe().transpose().drop("count",axis=1)
```

Out[12]:

	mean	std	min	25%	50%	75%	max
id	36517.829354	21161.721625	67.00	17741.250	36932.000	54682.00	72940.00
age	43.226614	22.612647	0.08	25.000	45.000	61.00	82.00
hypertension	0.097456	0.296607	0.00	0.000	0.000	0.00	1.00
heart_disease	0.054012	0.226063	0.00	0.000	0.000	0.00	1.00
avg_glucose_level	106.147677	45.283560	55.12	77.245	91.885	114.09	271.74
bmi	28.893237	7.854067	10.30	23.500	28.100	33.10	97.60
stroke	0.048728	0.215320	0.00	0.000	0.000	0.00	1.00

Data Cleaning

In [13]:

```
# check/drop null values
df.isnull().sum()
```

Out[13]:

```
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 [14]:

```
len(df)
```

Out[14]:

```
5110
```

In [15]:

```
df.isnull().sum()/len(df)
```

Out[15]:

```
id                   0.000000
gender               0.000000
age                  0.000000
hypertension         0.000000
heart_disease        0.000000
ever_married         0.000000
work_type            0.000000
Residence_type       0.000000
avg_glucose_level    0.000000
bmi                  0.039335
smoking_status       0.000000
stroke               0.000000
dtype: float64
```

In [16]:

```
df
```

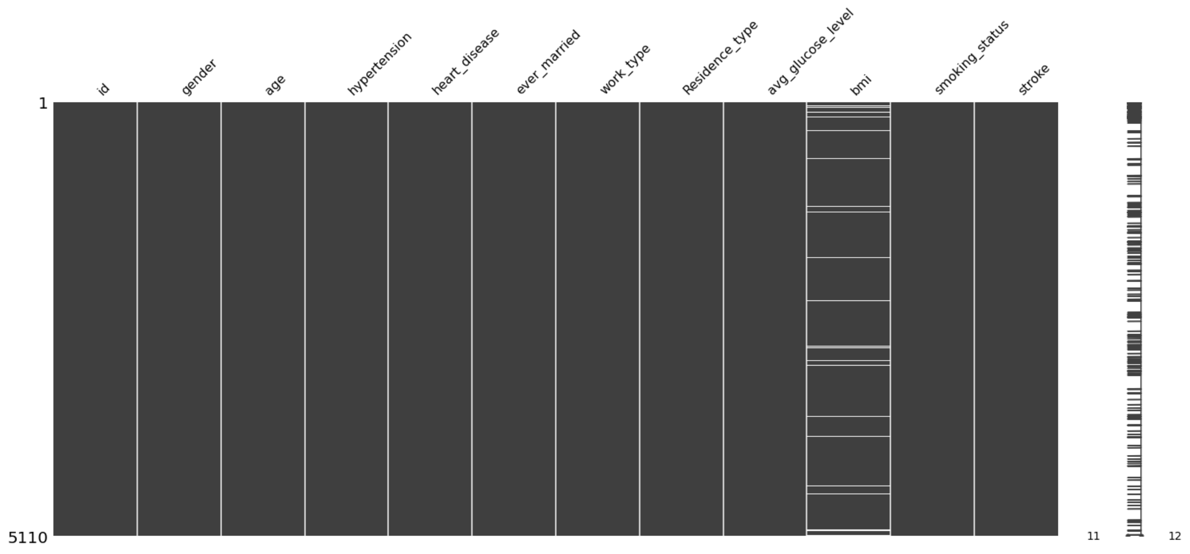
Out[16]:

	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
...	...	...	...	...	...	...	...	
5105	18234	Female	80.0	1	0	Yes	Private	Urb
5106	44873	Female	81.0	0	0	Yes	Self-employed	Urb
5107	19723	Female	35.0	0	0	Yes	Self-employed	Ru
5108	37544	Male	51.0	0	0	Yes	Private	Ru
5109	44679	Female	44.0	0	0	Yes	Govt_job	Urb

5110 rows × 12 columns

In [17]:

```
import missingno as msno
msno.matrix(df);
```



In [18]:

```
# df["bmi"] = df["bmi"].interpolate(method = "linear")
```

In [19]:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeRegressor
DT_bmi_pipe = Pipeline( steps=[
                               ('scale',StandardScaler()),
                               ('lr',DecisionTreeRegressor(random_state=42))
                              ])
X = df[['age','gender','bmi']].copy()
X.gender = X.gender.replace({'Male':0,'Female':1,'Other':-1}).astype(np.uint8)

Missing = X[X.bmi.isna()]
X = X[~X.bmi.isna()]
y = X.pop('bmi')
DT_bmi_pipe.fit(X,y)
predicted_bmi = pd.Series(DT_bmi_pipe.predict(Missing[['age','gender']]),index=Missing.index)
df.loc[Missing.index,'bmi'] = predicted_bmi
```

In [20]:

```
df.isnull().sum()
```

Out[20]:

```
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                  0
smoking_status       0
stroke               0
dtype: int64
```

In [21]:

```
# check blank
" " in df.values
```

Out[21]:

```
False
```

In [22]:

```
# check/drop duplicates
df[df.duplicated("id")]
```

Out[22]:

id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_

In [23]:

```
df = df[df["gender"]!="Other"]
```

In [24]:

```
df['bmi_cat'] = pd.cut(df['bmi'], bins = [0, 19, 25,30,10000], labels = ['Underweight', 'Ideal', 'Ov
```

In [25]:

```
df.bmi_cat.value_counts()
```

Out[25]:

```
Obesity        2011
Overweight     1475
Ideal          1203
Underweight     420
Name: bmi_cat, dtype: int64
```

In [26]:

```
## bin the family size.
def bmi_group(value):
    """
    This funciton create bmi groups(categories)
    """
    # result = ''
    if (value <= 19):
        result = 'Underweight'
    elif (value > 19 and value <= 25):
        result = 'Ideal'
    elif (value > 25 and value <= 30):
        result = 'Overweight'
    else:
        result = 'Obesity'
    return result
```

In [27]:

```
df["bmi_cat2"] = df["bmi"].apply(bmi_group)
df.bmi_cat2.value_counts()
```

Out[27]:

```
Obesity        2011
Overweight     1475
Ideal          1203
Underweight     420
Name: bmi_cat2, dtype: int64
```

In [28]:

```
df['bmi_cat3'] = df['bmi_cat2'].map( {'Underweight': 0,
                                      'Ideal': 1,
                                      'Overweight': 2,
                                      'Obesity': 3} ).astype(int)
```

In [29]:

```
df
```

Out[29]:

	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
...	...	...	...	...	...	...	...	
5105	18234	Female	80.0	1	0	Yes	Private	Urb
5106	44873	Female	81.0	0	0	Yes	Self-employed	Urb
5107	19723	Female	35.0	0	0	Yes	Self-employed	Ru
5108	37544	Male	51.0	0	0	Yes	Private	Ru
5109	44679	Female	44.0	0	0	Yes	Govt_job	Urb

5109 rows × 15 columns

EDA

stroke overview

In [30]:

```
stroke_summary = df.groupby("stroke").mean().reset_index()
stroke_summary
```

Out[30]:

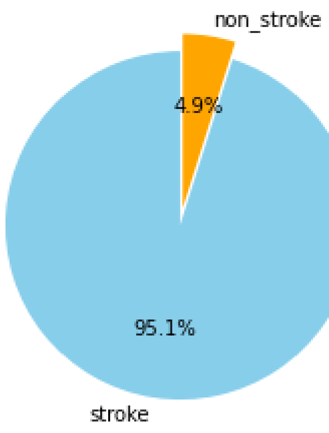
	stroke	id	age	hypertension	heart_disease	avg_glucose_level	bmi
0	0	36483.189300	41.974831	0.088889	0.047119	104.787584	28.847094
1	1	37115.068273	67.728193	0.265060	0.188755	132.544739	30.336552

In [31]:

```
stroke = df[df["stroke"]==1]
non_stroke = df[df["stroke"]==0]
```

In [32]:

```
labels = ["stroke","non_stroke"]
sizes = df.stroke.value_counts().values
plt.pie(x = sizes,labels=labels,explode=[0,0.1],startangle=90,colors = ['skyblue','orange'],autopct

# patches, texts,autotexts = plt.pie(sizes, labels=labels, colors=['skyblue','salmon'], explode=[0,0
```



In [16]:

```
df.stroke.value_counts(normalize=True)
```

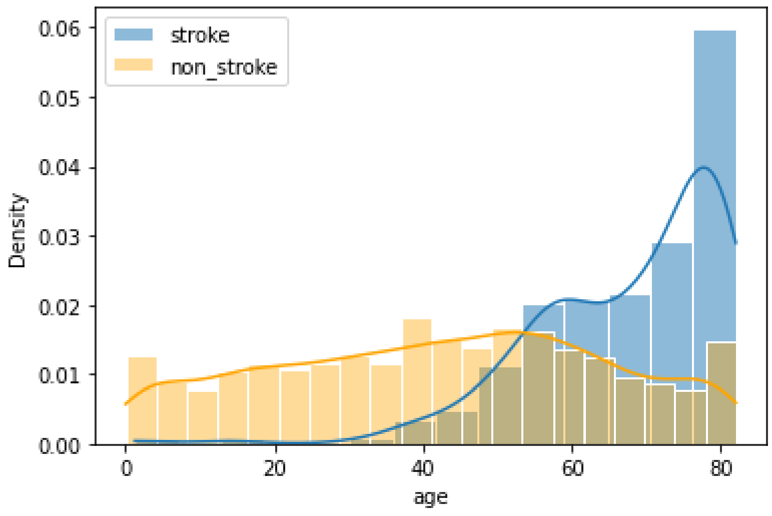
Out[16]:

```
0    0.951262
1    0.048738
Name: stroke, dtype: float64
```

Age

In [17]:

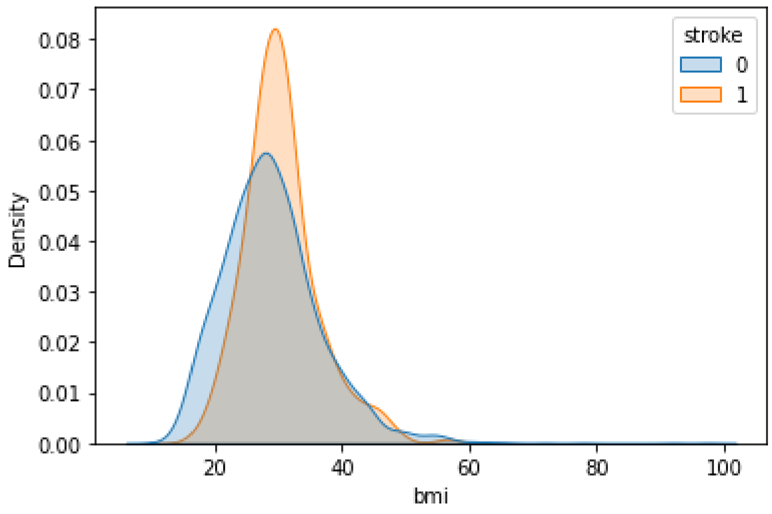
```
sns.histplot(stroke["age"],stat="density",kde=True,ec="w",label="stroke");
sns.histplot(non_stroke["age"],stat="density",kde=True,color="orange",ec="w",alpha=0.4,label="non_s
plt.legend(loc="best");
```



BMI

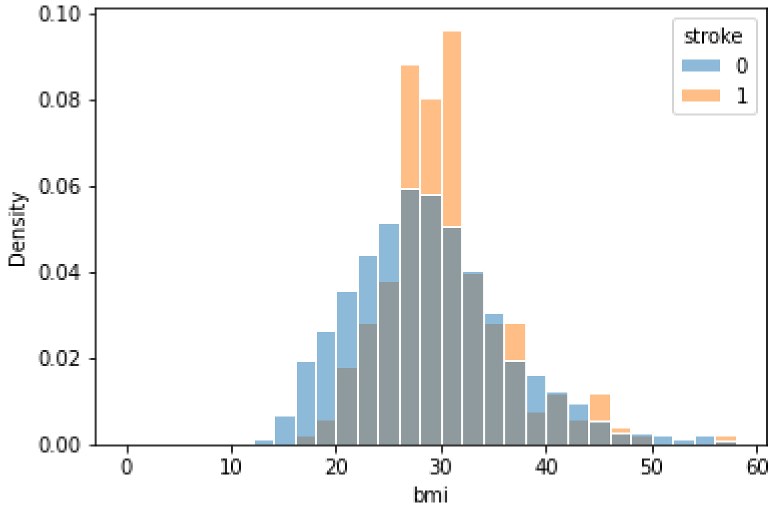
In [18]:

```
sns.kdeplot(data=df, x="bmi", hue="stroke",fill=True, common_norm=False);
```



In [19]:

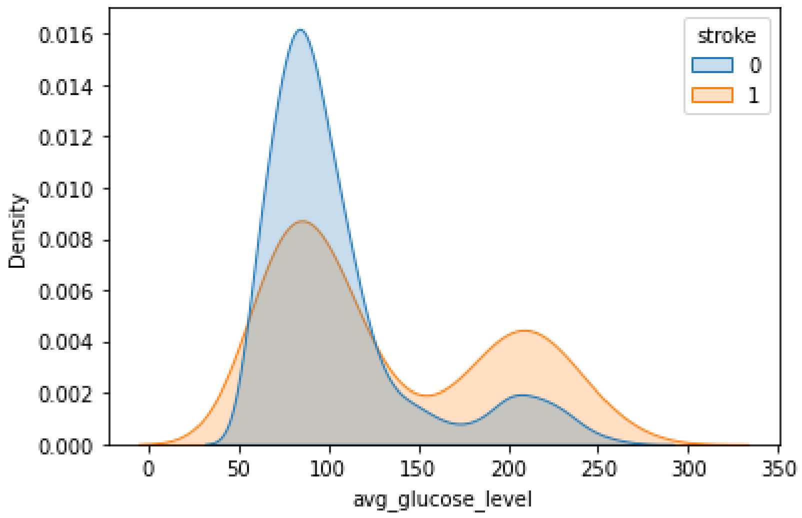
```
sns.histplot(data=df, x="bmi", hue="stroke",bins=np.arange(0,60,2),common_norm=False,stat="density
```



avg_glucose_level

In [20]:

```
sns.kdeplot(data=df, x="avg_glucose_level", hue="stroke",fill=True, common_norm=False);

# why valley>
```



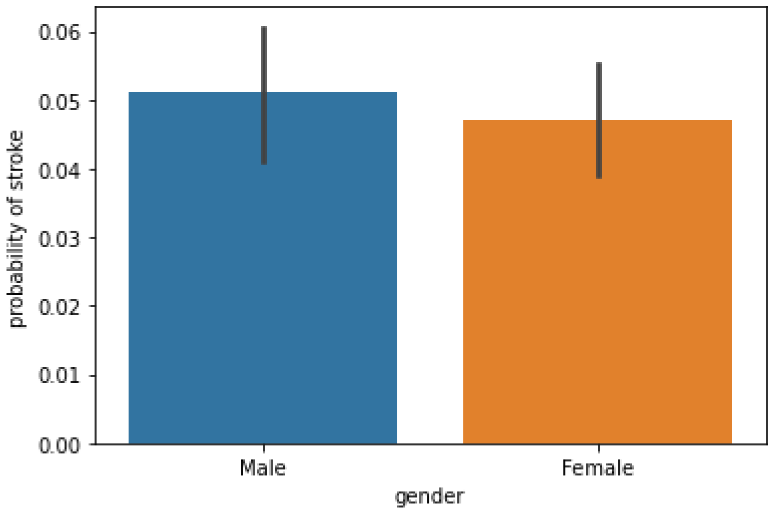
In [21]:

```
# df['bmi_cat'] = pd.cut(df['bmi'], bins = [0, 19, 25,30,10000], labels = ['Underweight', 'Ideal', '
# df['age_cat'] = pd.cut(df['age'], bins = [0,13,18, 45,60,200], labels = ['Children', 'Teens', 'Adu
# df['glucose_cat'] = pd.cut(df['avg_glucose_level'], bins = [0,90,160,230,500], labels = ['Low', 'N
```

gender

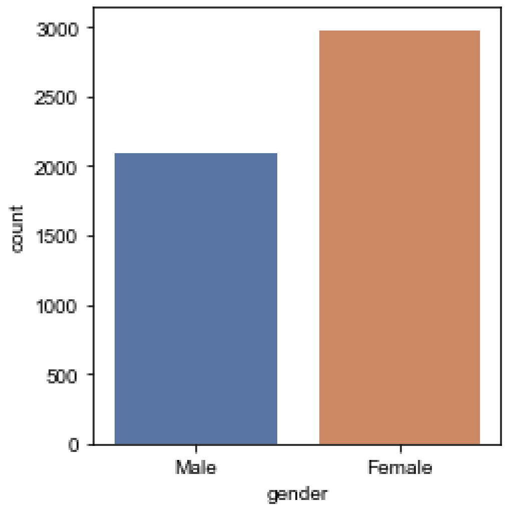
In [22]:

```
# 最重要的还是用barplot看比例
sns.barplot(data=df,x="gender",y="stroke")
plt.ylabel("probability of stroke", fontsize = 10);
```



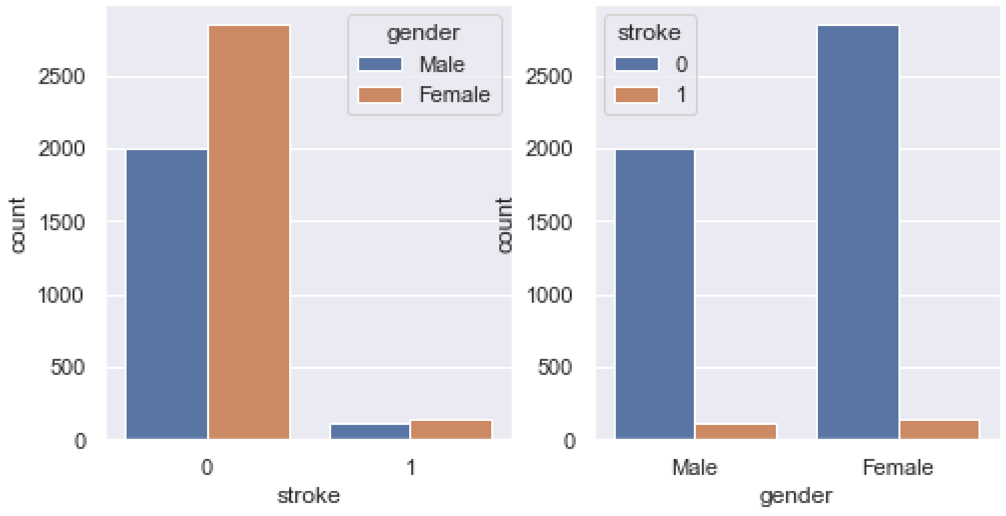
In [23]:

```
# 看一下数据中的男女数量分布，其实没什么用，重要的是看男女之间stroke的比例
plt.figure(figsize=(8,4))
plt.subplot(1,2,1)
sns.set(rc = {'figure.figsize':(5,4)})
sns.countplot(x='gender',data=df);
```



In [24]:

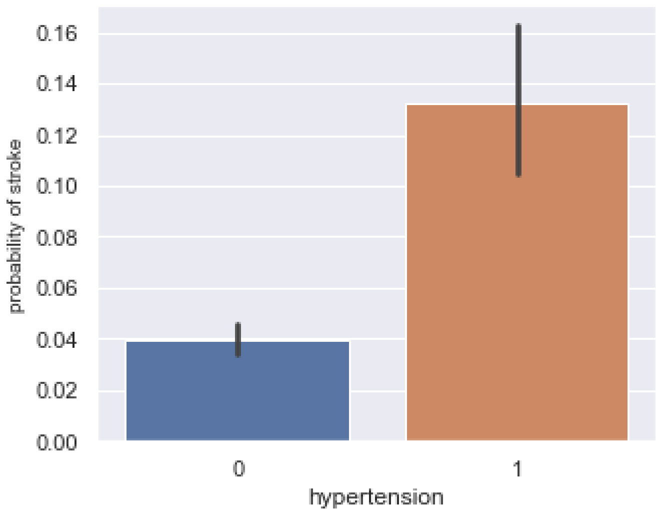
```
# 对于特别imbalanced的data 没什么用
plt.figure(figsize=(8,4))
plt.subplot(1,2,1)
sns.set(rc = {'figure.figsize':(5,4)})
sns.countplot(x='stroke',hue='gender',data=df);

plt.subplot(1,2,2)
sns.set(rc = {'figure.figsize':(5,4)})
sns.countplot(x='gender',hue='stroke',data=df);
```



hypertension

In [25]:

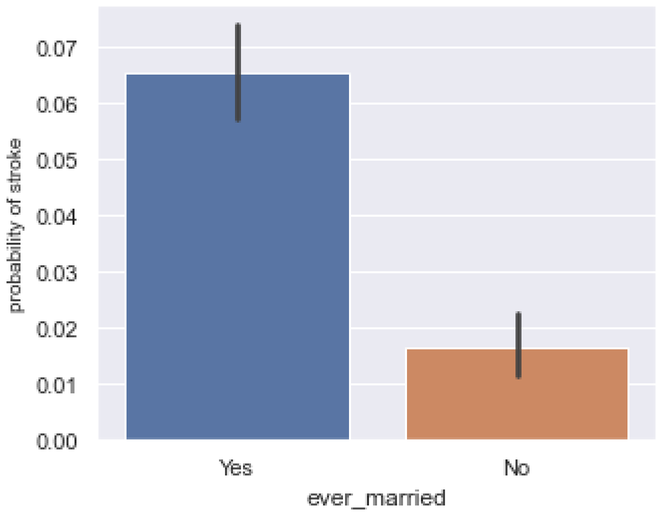
```
# 最重要的还是用barplot看比例
plt.figure(figsize=(5,4))
sns.barplot(data=df,x="hypertension",y="stroke")
plt.ylabel("probability of stroke", fontsize = 10);
```



ever married

In [26]:

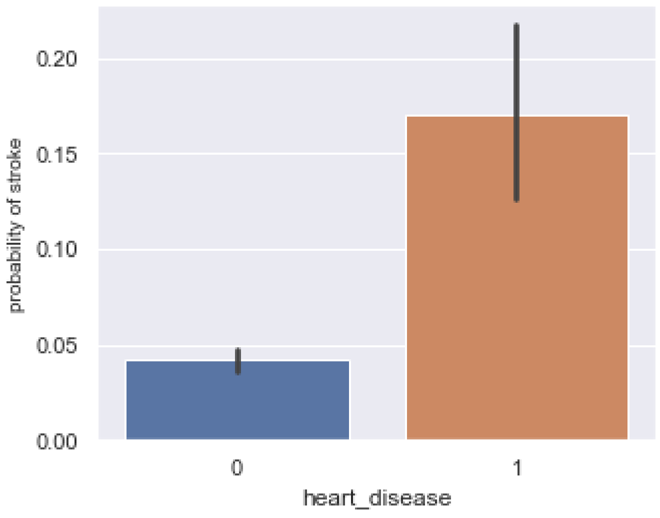
```
# 最重要的还是用barplot看比例
plt.figure(figsize=(5,4))
sns.barplot(data=df,x="ever_married",y="stroke")
plt.ylabel("probability of stroke", fontsize = 10);
```



heart disease

In [27]:

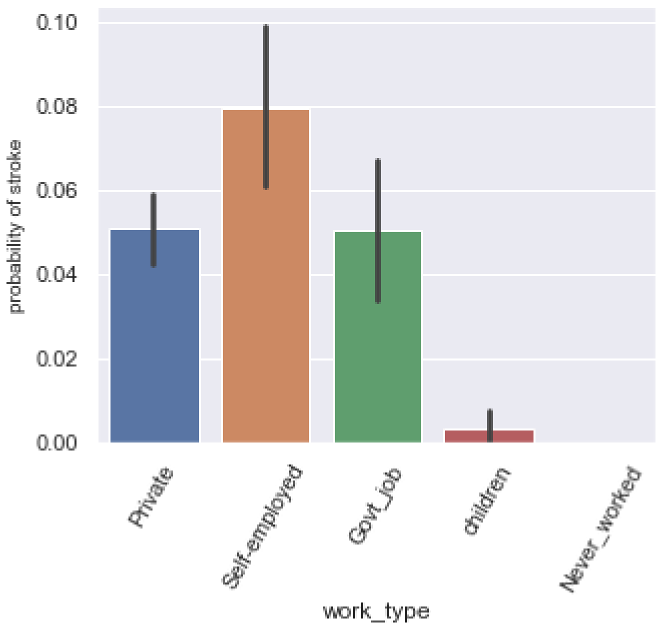
```
plt.figure(figsize=(5,4))
sns.barplot(data=df,x="heart_disease",y="stroke")
plt.ylabel("probability of stroke", fontsize = 10);
```



work type

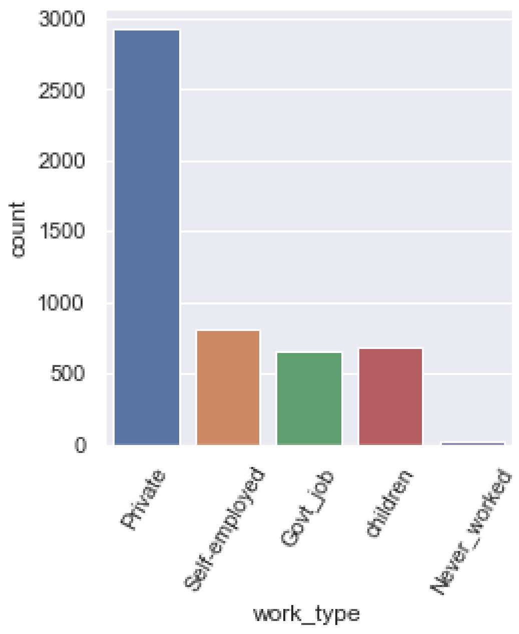
In [28]:

```
plt.figure(figsize=(5,4))
sns.barplot(data=df,x="work_type",y="stroke")
plt.xticks(rotation=60);
plt.ylabel("probability of stroke", fontsize = 10);
```



In [29]:

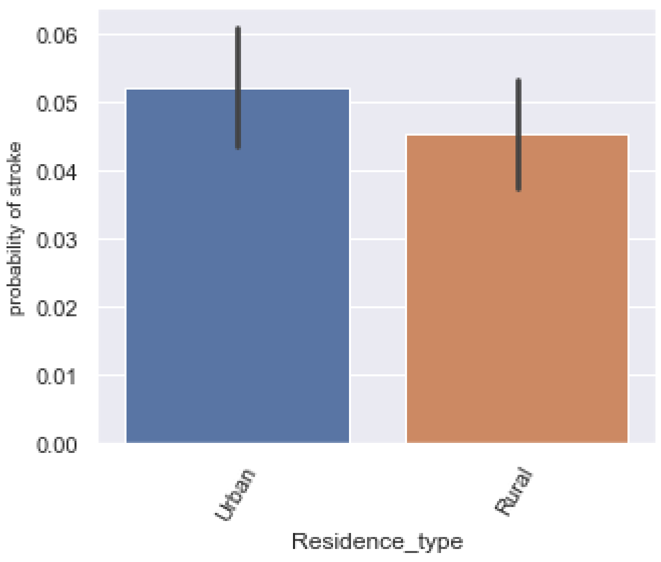
```
# 看一下数据中的work type数量分布，其实没什么用，重要的是看男女之间stroke的比例
plt.figure(figsize=(8,4))
plt.subplot(1,2,1)
sns.set(rc = {'figure.figsize':(5,4)})
plt.xticks(rotation=60)
sns.countplot(x='work_type',data=df);
```



residence type

In [30]:

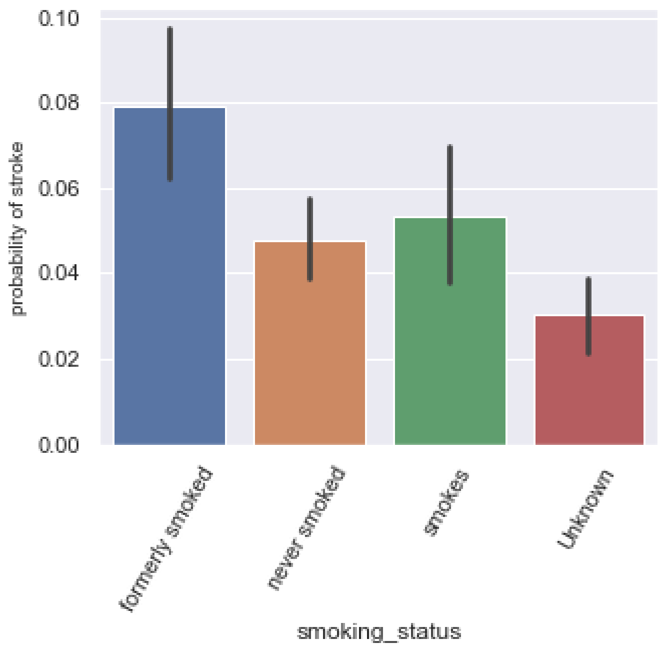
```
plt.figure(figsize=(5,4))
sns.barplot(data=df,x="Residence_type",y="stroke")
plt.xticks(rotation=60);
plt.ylabel("probability of stroke", fontsize = 10);
```



smoking status

In [31]:

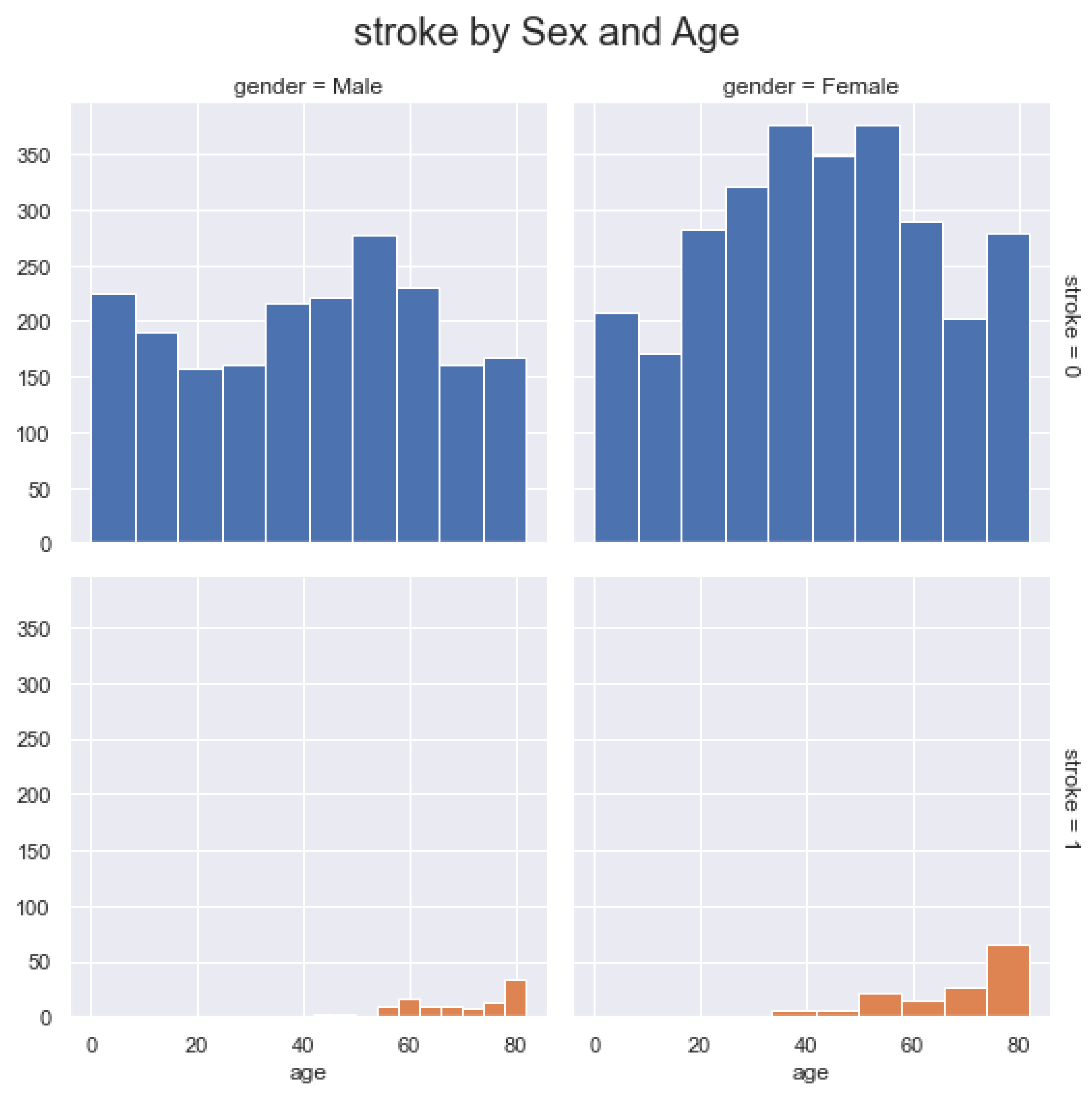
```
plt.figure(figsize=(5,4))
sns.barplot(data=df,x="smoking_status",y="stroke")
plt.xticks(rotation=60);
plt.ylabel("probability of stroke", fontsize = 10);
```



two dimenstion plot

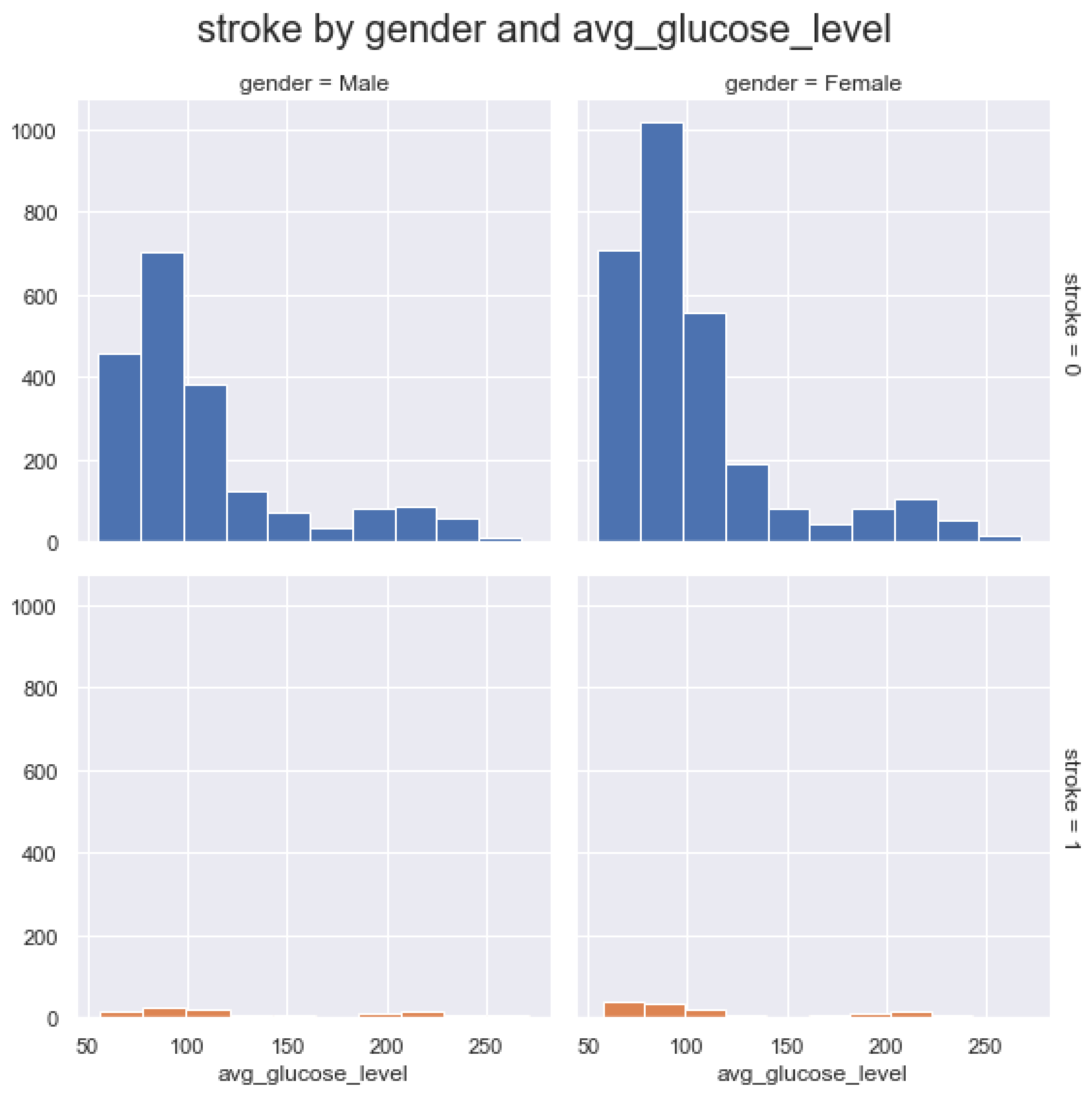
In [32]:

```
g = sns.FacetGrid(df,col="gender",row="stroke",hue = "stroke", margin_titles=True, size=4)
g = g.map(plt.hist, "age", edgecolor = 'white');
g.fig.suptitle("stroke by Sex and Age", size = 20)
plt.subplots_adjust(top=0.90);
```



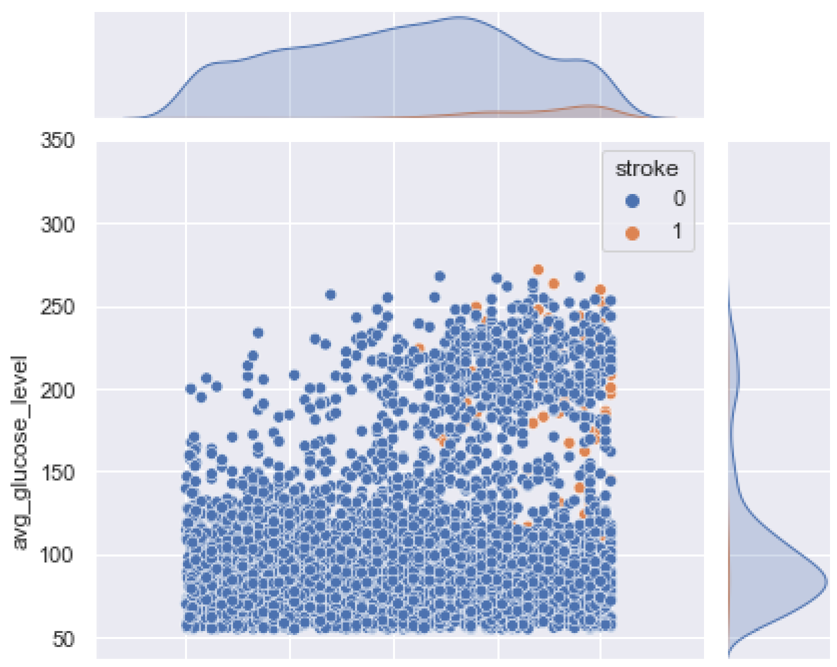
In [33]:

```
g = sns.FacetGrid(df,col="gender",row="stroke",hue = "stroke", margin_titles=True, size=4)
g = g.map(plt.hist, "avg_glucose_level", edgecolor = 'white');
g.fig.suptitle("stroke by gender and avg_glucose_level", size = 20)
plt.subplots_adjust(top=0.90);
```



In [34]:

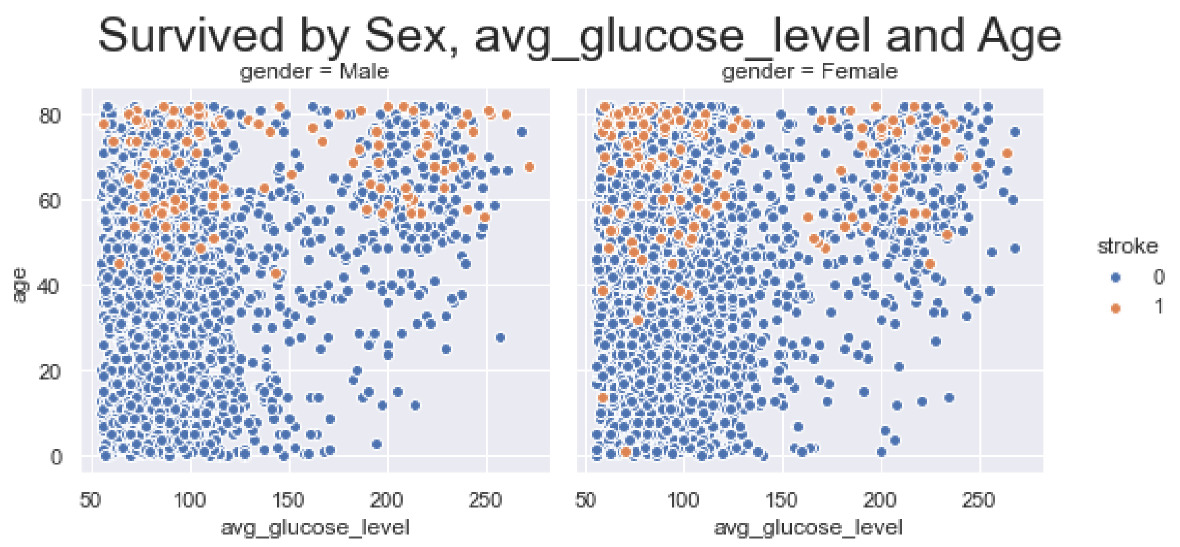
```
sns.jointplot(data=df,x="age",y="avg_glucose_level",hue="stroke");
```



three dimension plot

In [35]:

```
g = sns.FacetGrid(data=df,col="gender",hue="stroke",margin_titles=True,size = 4)
g.map(plt.scatter, "avg_glucose_level", "age",edgecolor="w").add_legend()
g.fig.suptitle("Survived by Sex, avg_glucose_level and Age", size = 25)
plt.subplots_adjust(top=0.85)

# The grid above clearly demonstrates the three outliers with Fare of over $500.
# At this point, I think we are quite confident that these outliers should be deleted.
# Most of the passengers were with in the Fare range of $100.
```



In [36]:

```
import plotly.express as px
fig = px.parallel_categories(df[['gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
       'work_type', 'Residence_type',
       'smoking_status', 'stroke']], color='stroke', color_continuous_scale=px.colors.sequential.Inf
fig.show()
```

Feature Engineering

In [37]:

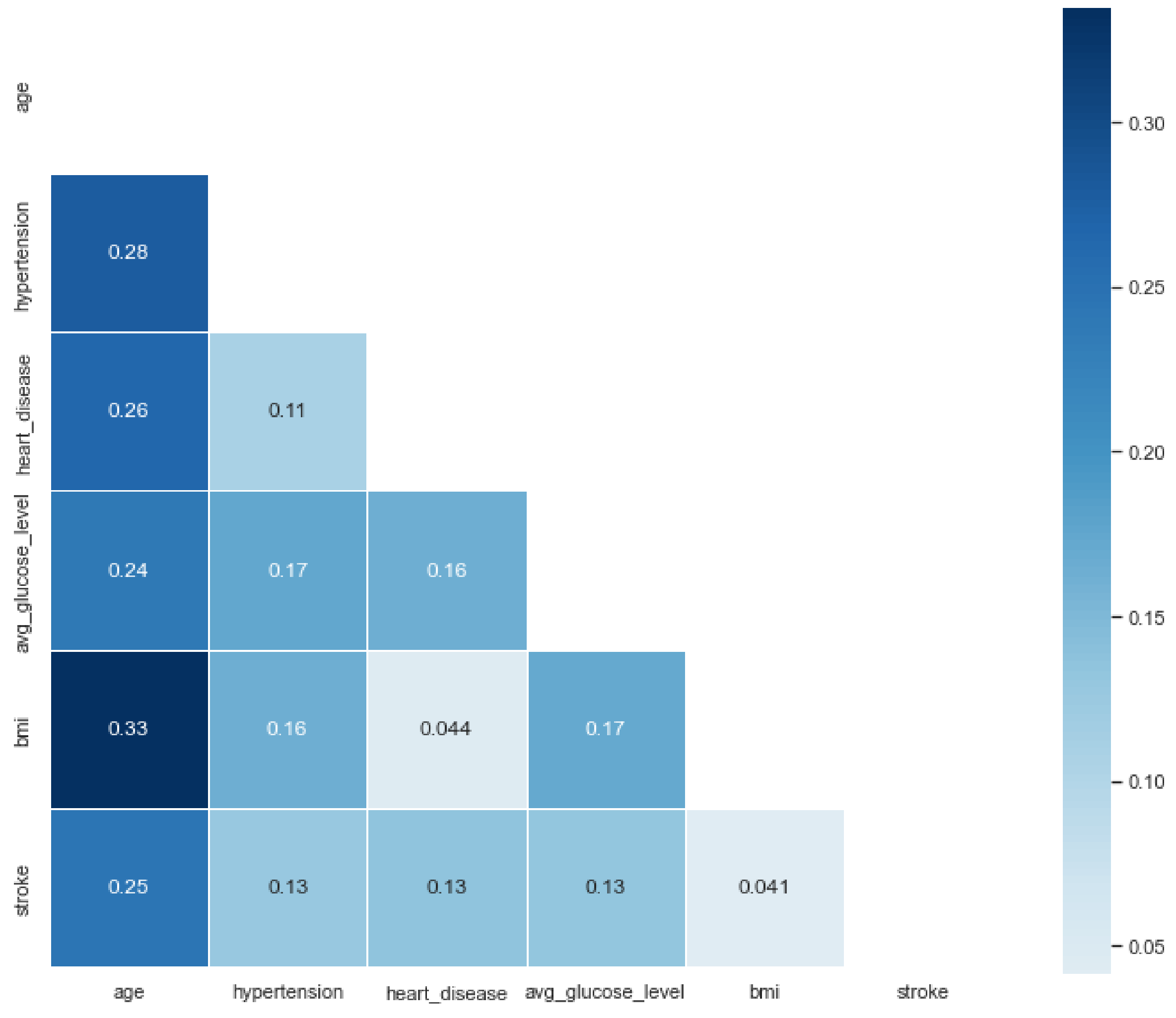
```
df = df.drop("id",axis=1)
```

In [38]:

```
X, y = df.drop("stroke",axis=1), df["stroke"]
```

In [39]:

```
mask = np.zeros_like(X.join(y).corr(), dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
sns.set_style('whitegrid')
plt.subplots(figsize = (12,10))
sns.heatmap(X.join(y).corr(),
            annot=True,       # 方块上显示数字correlation
            mask = mask,      # 为了弄一半
            cmap = 'RdBu',    # in order to reverse the bar replace "RdBu" with "RdBu_r"
            linewidths=.9,    # 方块间留点间隙
            linecolor='white',# 间隙的颜色弄白，默认好像就白色
            fmt='.2g',        # 可以不加
            center = 0,       # 最好加上，如果不加这个，就得用 vmin=-1, vmax=1,
            square=True);

# Already remove multicollinearity: SibSp and Parch
```



One hot encoding

In [40]:

```
# get dummies
X = pd.get_dummies(X,drop_first=True)
X
```

Out[40]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	gender_Male	ever_marri
0	67.0	0	1	228.69	36.600000	1	
1	61.0	0	0	202.21	29.879487	0	
2	80.0	0	1	105.92	32.500000	1	
3	49.0	0	0	171.23	34.400000	0	
4	79.0	1	0	174.12	24.000000	0	
...	...	...	...	...	...	...	
5105	80.0	1	0	83.75	28.476923	0	
5106	81.0	0	0	125.20	40.000000	0	
5107	35.0	0	0	82.99	30.600000	0	
5108	51.0	0	0	166.29	25.600000	1	
5109	44.0	0	0	85.28	26.200000	0	

5109 rows × 15 columns

Modeling

train and split

In [49]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.8,test_size=0.2,stratify=y,ran
```

In [50]:

```
X_train
```

Out[50]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	gender_Male	ever_married_Ye
845	48.0	0	0	69.21	33.1	0	
3745	29.0	0	0	84.19	21.2	0	
4184	35.0	0	0	119.40	22.9	0	
3410	38.0	0	0	108.68	32.7	1	
284	14.0	0	0	82.34	31.6	1	
...	...	...	...	...	...	...	
1434	45.0	0	0	92.86	35.1	0	
461	16.0	0	0	113.47	19.5	0	
1052	61.0	0	0	78.65	36.2	0	
1757	31.0	0	0	74.05	26.0	1	
5053	46.0	0	0	55.84	27.8	0	

4087 rows × 15 columns

Scaling (standardization)

In [51]:

```
# from sklearn.preprocessing import StandardScaler
# scaler = StandardScaler()
# X_train_std = scaler.fit_transform(X_train)
# X_test_std = scaler.transform(X_test)
```

In [52]:

```
# X_train_std_df = pd.DataFrame(data=X_train_std, columns=X_train.columns)
# X_test_std_df = pd.DataFrame(data=X_test_std, columns=X_test.columns)
```

oversampling SMOTE

In [53]:

```
print("Before OverSampling, counts of label '1': {}".format(sum(y_train==1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train==0)))

from imblearn.over_sampling import SMOTE
oversample = SMOTE("minority")
X_train,y_train = oversample.fit_resample(X_train,y_train)

print("After OverSampling, counts of label '1': {}".format(sum(y_train==1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train==0)))
```

```
Before OverSampling, counts of label '1': 199
Before OverSampling, counts of label '0': 3888

After OverSampling, counts of label '1': 3888
After OverSampling, counts of label '0': 3888
```

RF

In [54]:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train,y_train)
y_pred = rf.predict(X_test)
```

In [55]:

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

Out[55]:

```
0.9129158512720157
```

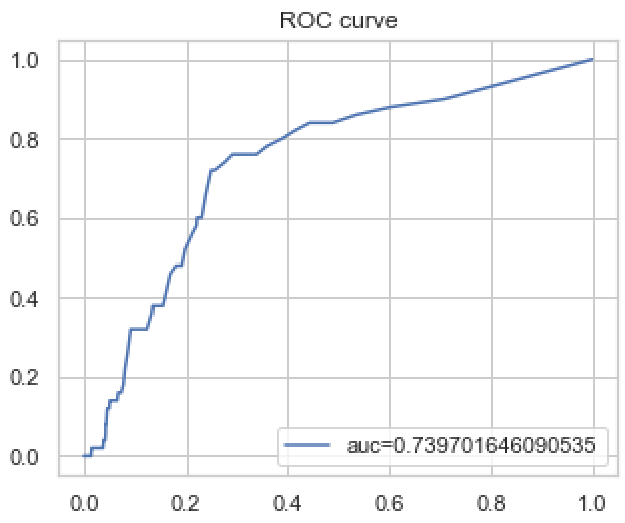
In [56]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.95      0.95       972
           1       0.12      0.12      0.12        50

    accuracy                           0.91      1022
   macro avg       0.54      0.54      0.54      1022
weighted avg       0.91      0.91      0.91      1022
```

In [57]:

```
from sklearn import metrics
from sklearn.metrics import roc_auc_score
y_pred_proba = rf.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,y_pred_proba)

auc = metrics.roc_auc_score(y_test,y_pred_proba)
plt.plot(fpr,tpr,label = "auc="+str(auc))
plt.legend(loc=4)
plt.title("ROC curve")
plt.show()
```



LR

In [58]:

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(solver='liblinear',penalty= 'l1',random_state = 42)

## fit the model with "train_x" and "train_y"
logreg.fit(X_train,y_train)

## Once the model is trained we want to find out how well the model is performing, so we test the mo
## we use "X_test" portion of the data(this data was not used to fit the model) to predict model out
y_pred = logreg.predict(X_test)
```

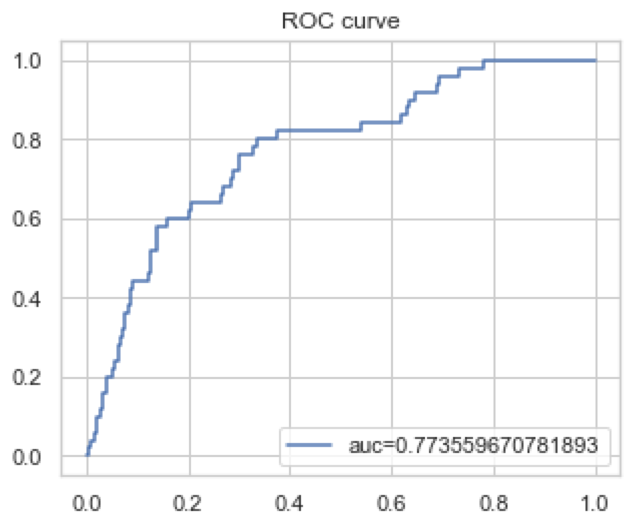
In [59]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.84      0.90       972
           1       0.16      0.58      0.25        50

    accuracy                           0.83      1022
   macro avg       0.57      0.71      0.58      1022
weighted avg       0.94      0.83      0.87      1022
```

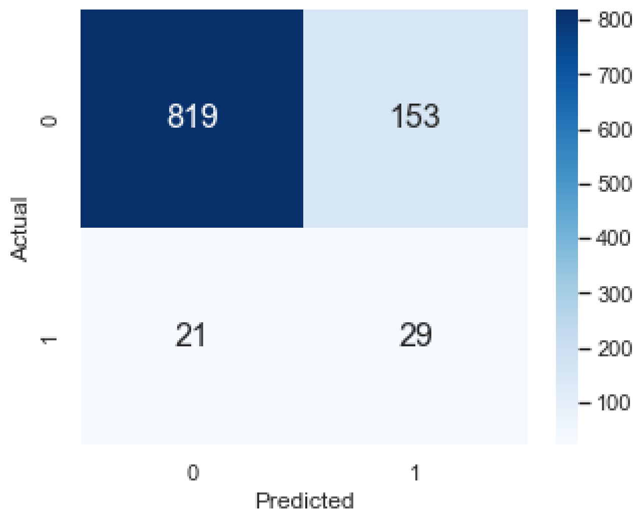
In [60]:

```
from sklearn import metrics
y_pred_proba = logreg.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,y_pred_proba)

auc = metrics.roc_auc_score(y_test,y_pred_proba)
plt.plot(fpr,tpr,label = "auc="+str(auc))
plt.legend(loc=4)
plt.title("ROC curve")
plt.show()
```



In [61]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt = 'd', cmap = 'Blues', annot_kws = {'size': 16})
plt.xlabel('Predicted')
plt.ylabel('Actual');
```



In [62]:

```
from sklearn.metrics import f1_score
f1_score(y_test, y_pred, average='weighted')
```

Out[62]:

```
0.871978719819598
```

In [63]:

```
f1_score(y_test, y_pred)
```

Out[63]:

```
0.25
```

In [64]:

```
f1_score(y_test, y_pred, average='macro')
```

Out[64]:

```
0.5769867549668874
```

Model Explanability

In [65]:

```
X_train
```

Out[65]:

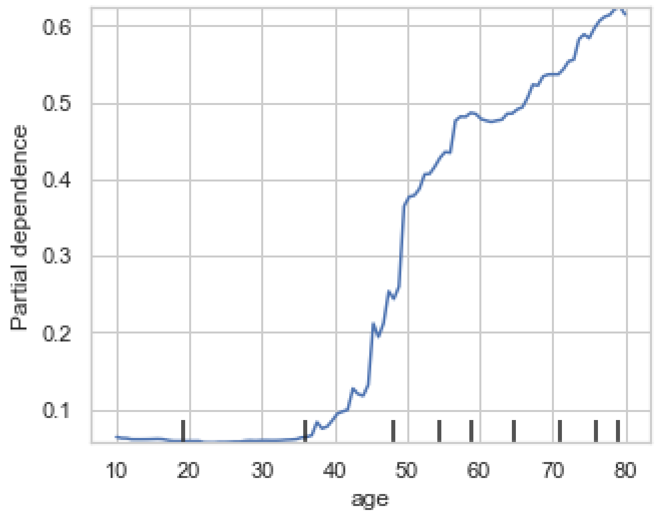
	age	hypertension	heart_disease	avg_glucose_level	bmi	gender_Male	ever_married_Yes
0	48.000000	0	0	69.210000	33.100000	0	1
1	29.000000	0	0	84.190000	21.200000	0	0
2	35.000000	0	0	119.400000	22.900000	0	1
3	38.000000	0	0	108.680000	32.700000	1	1
4	14.000000	0	0	82.340000	31.600000	1	0
...	...	...	...	...	...	...	...
7771	59.333464	0	0	112.610615	32.800313	0	1
7772	57.000000	0	0	85.600565	34.309833	1	1
7773	57.608184	0	0	218.815632	35.675792	0	1

In [66]:

```
# feature_names = [i for i in X_train_std_df.columns if X_train_std_df[i].dtype in [np.float64]]
# target_names = "stroke"

# from sklearn import tree
# import graphviz
# from sklearn.tree import export_graphviz

# dot_data = export_graphviz(rf.estimators_[0],
#                              feature_names= feature_names,
#                              class_names= target_names,
#                              filled=True, impurity=True,
#                              rounded=True,out_file=None)

# graph = graphviz.Source(dot_data, format='png')
# graph
```

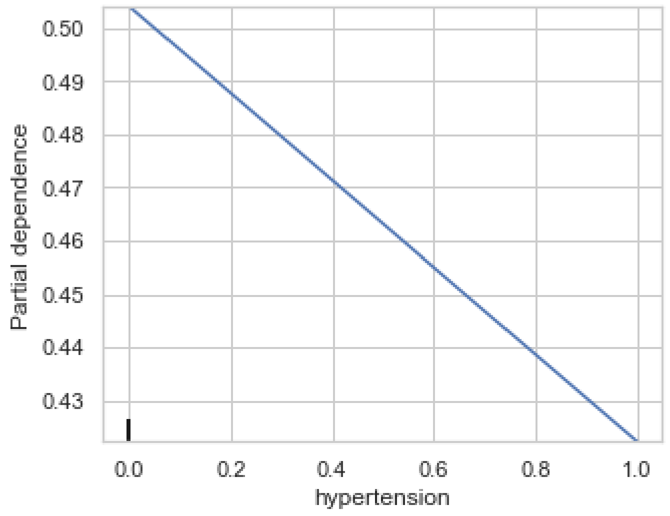
In [67]:

```
from sklearn.inspection import PartialDependenceDisplay

PartialDependenceDisplay.from_estimator(rf, X_train,['age'],kind='average');
```



In [68]:

```
PartialDependenceDisplay.from_estimator(rf, X_train,['hypertension'],kind='average');
```

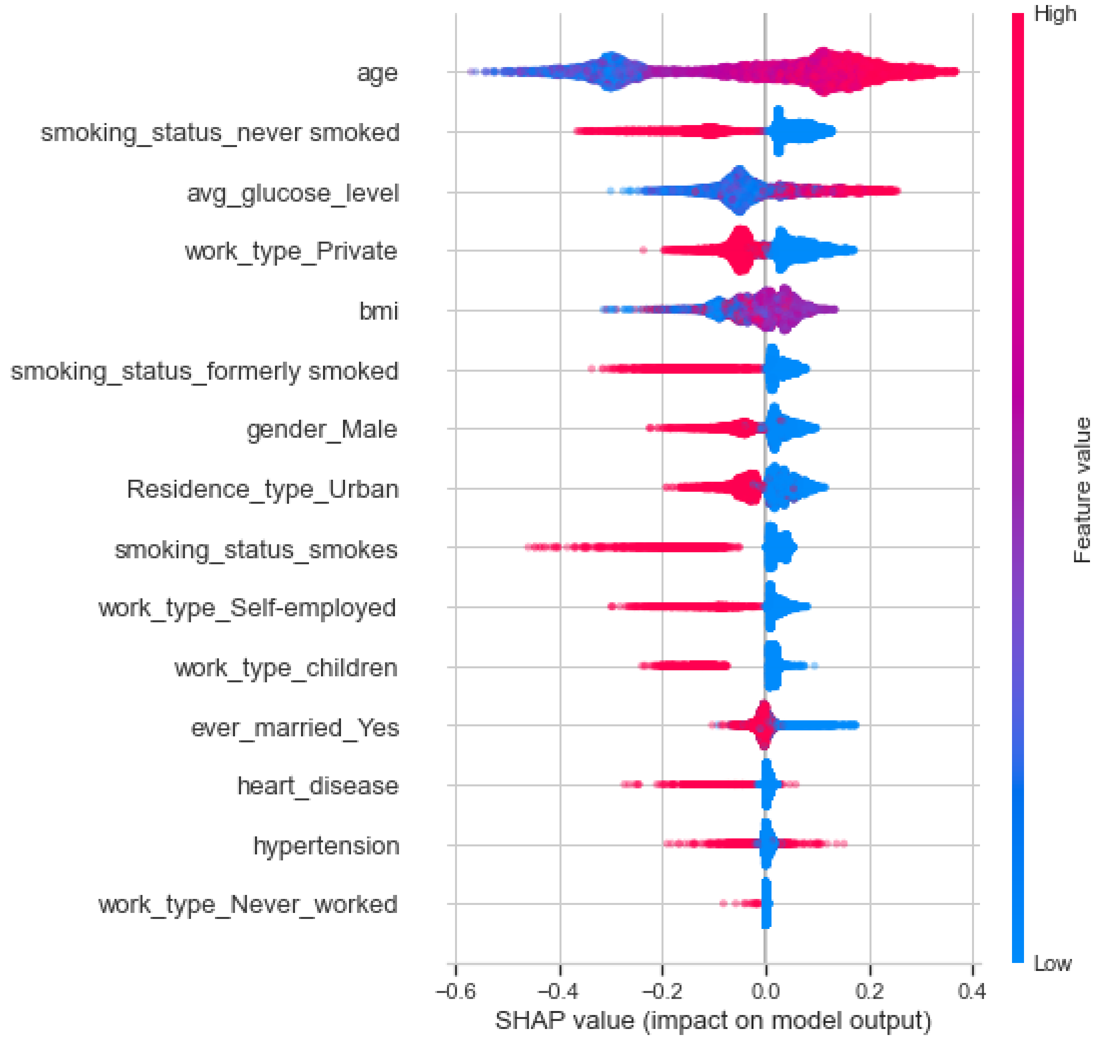


In [69]:

```
import shap

explainer = shap.TreeExplainer(rf)

# calculate shap values. This is what we will plot.
shap_values = explainer.shap_values(X_train)

# The shap_values object above is a list with two arrays. the second array is the list of SHAP value
# We typically think about predictions in terms of the prediction of a positive outcome
```

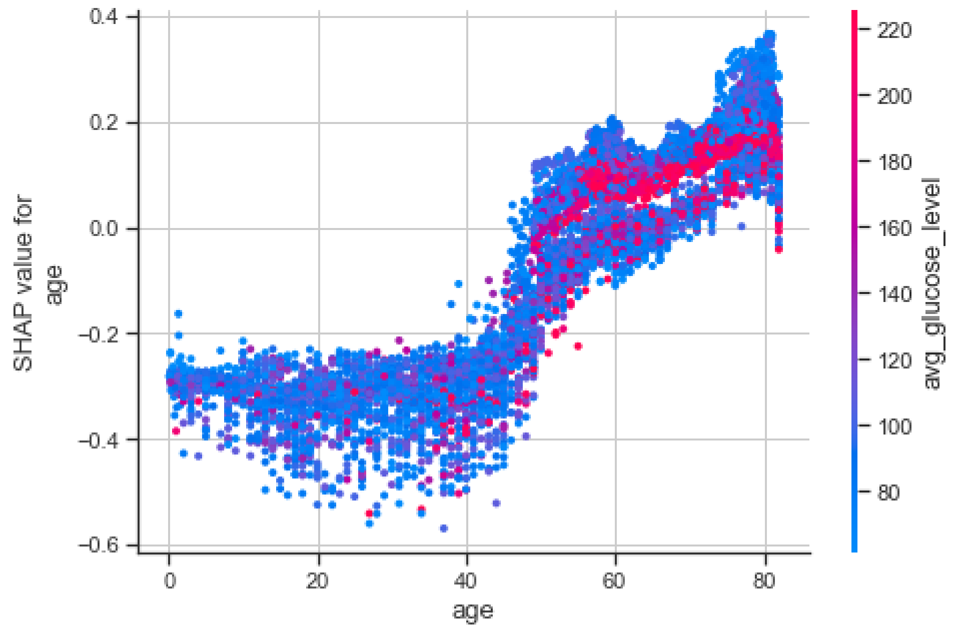
In [70]:

```
shap.summary_plot(shap_values[1], X_train,alpha=0.4)

# The summary plot combines feature importance with feature effects.
# Each point on the summary plot is a Shapley value for a feature and an instance.
# The position on the y-axis is determined by the feature and on the x-axis by the Shapley value.

# Feature importance: (Vertical location) Variables are ranked in descending order.
# Impact: (Horizontal location) shows whether the effect of that value is associated with a higher o
```



In [71]:

```
# The SHAP Dependence plot shows the marginal effect one or two features have on the predicted outco
# It tells whether the relationship between the target and a feature is linear, monotonic or more co
shap.dependence_plot('age', shap_values[1], X_train)
```



In [72]:

```
display(X_train.loc[[4082]])

choosen_instance = X_train.loc[[4082]]
shap_values = explainer.shap_values(choosen_instance)
shap.initjs()
shap.force_plot(explainer.expected_value[1], shap_values[1], choosen_instance)

# Feature values in pink cause to increase the prediction. Feature values in blue cause to decrease
# Size of the bar shows the magnitude of the feature's effect.
```

	age	hypertension	heart_disease	avg_glucose_level	bmi	gender_Male	ever_married_Ye
4082	45.0	0	0	92.86	35.1	0	

Out[72]:

Visualization omitted, Javascript library not loaded!
Have you run `initjs()` in this notebook? If this notebook was from another user you must also trust this notebook (File -> Trust notebook). If you are viewing this notebook on github the Javascript has been stripped for security. If you are using JupyterLab this error is because a JupyterLab extension has not yet been written.

In [ ]:

```

```

In [ ]:

```

```