Brain Stroke EDA and Prediction

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv("brain_stroke.csv")
print(f"\033[034m\033[1m")
print("Features Name :")
print(f"\033[031m\033[1m")
print(df.columns.tolist())
print(f"\033[035m\033[1m")
print("Data Shape :")
print(df.shape)

print(f"\033[034m\033[1m")
print("Data:")

df.head(10).style.set_properties(**{'background-color': 'Blue',
                           'color': 'white',
                           'border-color': 'darkblack'})
```

Features Name :

```
['gender', 'age', 'hypertension', 'heart_disease', 'ever_married', 'work_type', 'Residence_type', 'avg_glucose_level', 'bmi', 'smoking_status', 'stroke']
```

Data Shape :

```
(4981, 11)
```

Data:

Out[2]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
0	Male	67.000000	0	1	Yes	Private	Urban	228.690000	36.600000	formerly smoked
1	Male	80.000000	0	1	Yes	Private	Rural	105.920000	32.500000	never smoked
2	Female	49.000000	0	0	Yes	Private	Urban	171.230000	34.400000	smokes
3	Female	79.000000	1	0	Yes	Self-employed	Rural	174.120000	24.000000	never smoked
4	Male	81.000000	0	0	Yes	Private	Urban	186.210000	29.000000	formerly smoked
5	Male	74.000000	1	1	Yes	Private	Rural	70.090000	27.400000	never smoked
6	Female	69.000000	0	0	No	Private	Urban	94.390000	22.800000	never smoked
7	Female	78.000000	0	0	Yes	Private	Urban	58.570000	24.200000	Unknown
8	Female	81.000000	1	0	Yes	Private	Rural	80.430000	29.700000	never smoked
9	Female	61.000000	0	1	Yes	Govt_job	Rural	120.460000	36.800000	smokes

Attribute Information 1) gender: "Male", "Female" or "Other" 2) age: age of the patient 3) hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension 4) heart disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease 5) Ever-married: "No" or "Yes" 6) work type: "children", "Govtjov", "Never worked", "Private" or "Self-employed" 7) Residencetype: "Rural" or "Urban" 8) avg glucose level: average glucose level in blood 9) BMI: body mass index 10) smoking_status: "formerly smoked", "never smoked", "smokes" or "Unknown" *11) stroke: 1 if the patient had a stroke or 0 if not*Note: "Unknown" in smoking_status means tha

A stroke is a medical condition in which poor blood flow to the brain causes cell death. There are two main types of stroke: ischemic, due to lack of blood flow, and hemorrhagic, due to bleeding. Both cause parts of the brain to stop functioning properly. Signs and symptoms of a stroke may include an inability to move or feel on one side of the body, problems understanding or speaking, dizziness, or loss of vision

to one side. Signs and symptoms often appear soon after the stroke has occurred. If symptoms last less than one or two hours, the stroke is a transient ischemic attack (TIA), also called a mini-stroke. A hemorrhagic stroke may also be associated with a severe headache. The symptoms of a stroke can be permanent. Long-term complications may include pneumonia and loss of bladder control.

The main risk factor for stroke is high blood pressure. Other risk factors include high blood cholesterol, tobacco smoking, obesity, diabetes mellitus, a previous TIA, end-stage kidney disease, and atrial fibrillation. An ischemic stroke is typically caused by blockage of a blood vessel, though there are also less common causes. A hemorrhagic stroke is caused by either bleeding directly into the brain or into the space between the brain's membranes. Bleeding may occur due to a ruptured brain aneurysm. Diagnosis is typically based on a physical exam and is supported by medical imaging such as a CT scan or MRI scan. A CT scan can rule out bleeding, but may not necessarily rule out ischemia, which early on typically does not show up on a CT scan. Other tests such as an electrocardiogram (ECG) and blood tests are done to determine risk factors and rule out other possible causes. Low blood sugar may cause similar symptoms.

Prevention includes decreasing risk factors, surgery to open up the arteries to the brain in those with problematic carotid narrowing, and warfarin in people with atrial fibrillation. Aspirin or statins may be recommended by physicians for prevention. A stroke or TIA often requires emergency care. An ischemic stroke, if detected within three to four and half hours, may be treatable with a medication that can break down the clot. Some hemorrhagic strokes benefit from surgery. Treatment to attempt recovery of lost function is called stroke rehabilitation, and ideally takes place in a stroke unit; however, these are not available in much of the world.

Description of Stroke Data

In [3]:
```
df.describe().round(2).T.sort_values(by='std' , ascending = False)\
                    .style.background_gradient(cmap='GnBu')\
                    .bar(subset=["max"], color='#BB0000')\
                    .bar(subset=["min",], color='green')\
                    .bar(subset=["mean",], color='Orange')\
                    .bar(subset=['std'], color='pink')\
                    .bar(subset=['50%'], color='magenta')
```

Out[3]:

	count	mean	std	min	25%	50%	75%	max
avg_glucose_level	4981.000000	105.940000	45.080000	55.120000	77.230000	91.850000	113.860000	271.740000
age	4981.000000	4	22		25.000000	45	61.000000	
bmi	4962.000000				23.700000		32.600000	
hypertension	4981.000000			0.000000	0.000000	0.000000	0.000000	
heart_disease	4981.000000			0.000000	0.000000	0.000000	0.000000	
stroke	4981.000000	0.050000		0.000000	0.000000	0.000000	0.000000	

In [4]:
```
def summary(train):
    sum = pd.DataFrame(train.dtypes, columns=['dtypes'])
    sum['missing#'] = train.isna().sum()
    sum['missing%'] = (train.isna().sum())/len(train)
    sum['uniques'] = train.nunique().values
    sum['count'] = train.count().values
    return sum

summary(df).style.background_gradient(cmap='RdPu_r')
```

Out[4]:

	dtypes	missing#	missing%	uniques	count
gender	object	0	0.000000	2	4981
age	float64	0	0.000000	104	4981
hypertension	int64	0	0.000000	2	4981
heart_disease	int64	0	0.000000	2	4981
ever_married	object	0	0.000000	2	4981
work_type	object	0	0.000000	4	4981
Residence_type	object	0	0.000000	2	4981
avg_glucose_level	float64	0	0.000000	3895	4981
bmi	float64	19	0.003814	342	4962
smoking_status	object	0	0.000000	4	4981
stroke	int64	0	0.000000	2	4981

In [5]:
```
stroke = df[df['stroke'] == 1]
print(f"\033[031m\033[1m")
print("Data of Person with Brain Stroke :")
print(f"\033[035m\033[1m")
print("Data Shape :")
print(stroke.shape)

print(f"\033[034m\033[1m")
print("Data:")

stroke.head(5).style.set_properties(**{'background-color': 'Red',
                           'color': 'white',
                           'border-color': 'darkblack'})
```

Data of Person with Brain Stroke :

Data Shape :
(248, 11)

Data:

Out[5]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
0	Male	67.000000	0	1	Yes	Private	Urban	228.690000	36.600000	formerly smoked
1	Male	80.000000	0	1	Yes	Private	Rural	105.920000	32.500000	never smoked
2	Female	49.000000	0	0	Yes	Private	Urban	171.230000	34.400000	smokes
3	Female	79.000000	1	0	Yes	Self-employed	Rural	174.120000	24.000000	never smoked
4	Male	81.000000	0	0	Yes	Private	Urban	186.210000	29.000000	formerly smoked

In [6]:
```
HeartDisease = df[df['heart_disease'] == 1]
print(f"\033[031m\033[1m")
print("Data of Person with heart Disease :")
print(f"\033[035m\033[1m")
print("Data Shape :")
print(HeartDisease.shape)

print(f"\033[034m\033[1m")
print("Data:")

HeartDisease.head(5).style.set_properties(**{'background-color': 'Red',
                           'color': 'white',
                           'border-color': 'darkblack'})
```

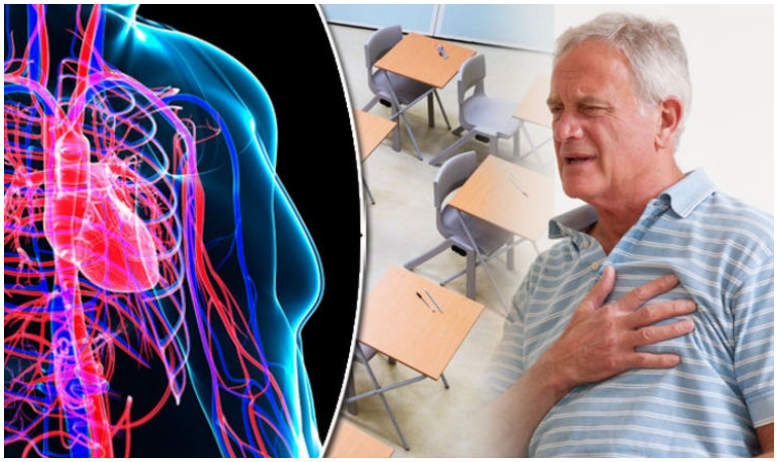
Data of Person with heart Disease :

Data Shape :
(275, 11)

Data:

Out[6]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
0	Male	67.000000	0	1	Yes	Private	Urban	228.690000	36.600000	formerly smoked
1	Male	80.000000	0	1	Yes	Private	Rural	105.920000	32.500000	never smoked
5	Male	74.000000	1	1	Yes	Private	Rural	70.090000	27.400000	never smoked
9	Female	61.000000	0	1	Yes	Govt_job	Rural	120.460000	36.800000	smokes
11	Female	79.000000	0	1	Yes	Private	Urban	214.090000	28.200000	never smoked



In [7]:
```
Healthy = df[df['heart_disease'] == 1]
print(f"\033[031m\033[1m")
print("Data of Person with No heart Disease and No Stroke:")
print(f"\033[035m\033[1m")
print("Data Shape :")
print(Healthy.shape)

print(f"\033[034m\033[1m")
print("Data:")

Healthy.head(5).style.set_properties(**{'background-color': 'Green',
                           'color': 'white',
                           'border-color': 'darkblack'})
```

```
Data of Person with No heart Disease and No Stroke:

Data Shape :
(275, 11)

Data:
```

Out[7]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
0	Male	67.000000	0	1	Yes	Private	Urban	228.690000	36.600000	formerly smoked
1	Male	80.000000	0	1	Yes	Private	Rural	105.920000	32.500000	never smoked
5	Male	74.000000	1	1	Yes	Private	Rural	70.090000	27.400000	never smoked
9	Female	61.000000	0	1	Yes	Govt_job	Rural	120.460000	36.800000	smokes
11	Female	79.000000	0	1	Yes	Private	Urban	214.090000	28.200000	never smoked

Data Information

In [8]:
```
print(f"\033[94m\033[1m")
print("Information of training dataset \n")
print(f"\033[91m\033[1m")
df.info()
```

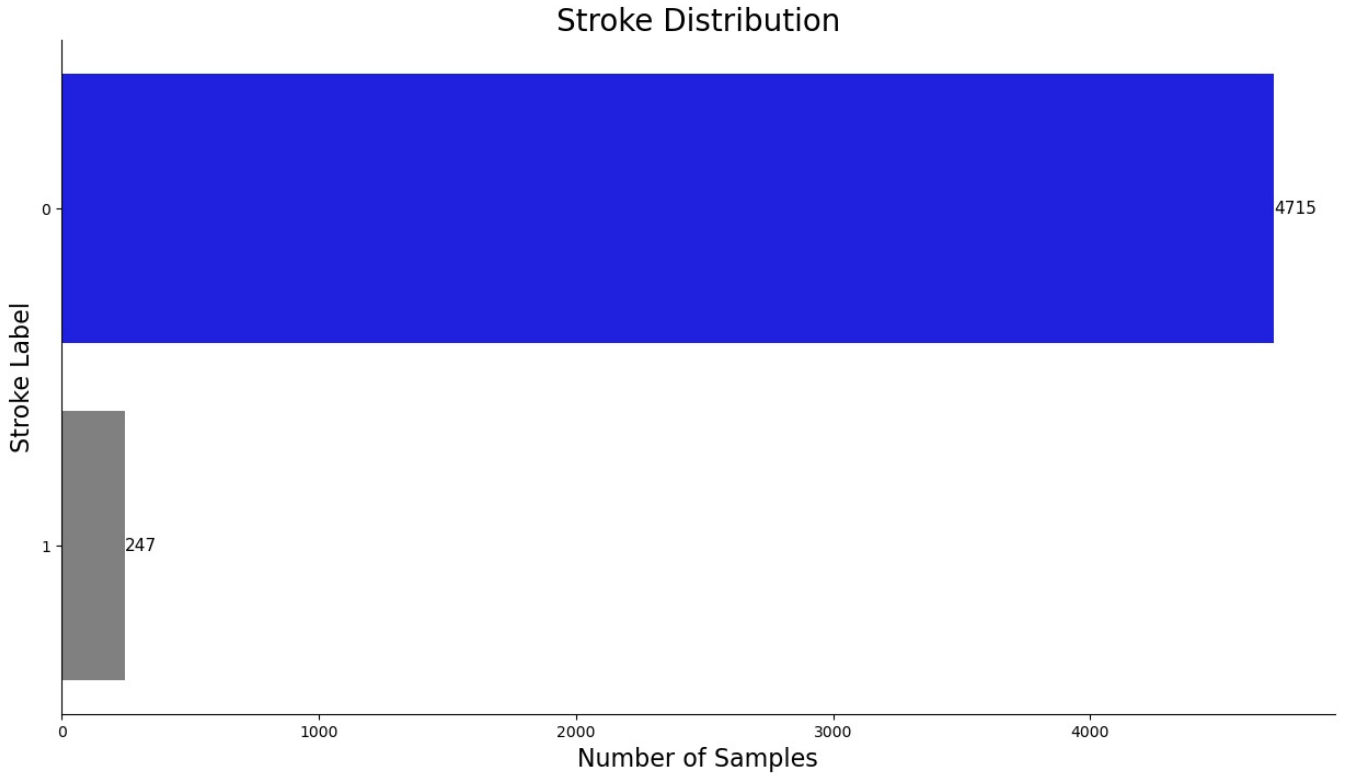
Information of training dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4981 entries, 0 to 4980
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   gender             4981 non-null   object 
 1   age                4981 non-null   float64
 2   hypertension       4981 non-null   int64  
 3   heart_disease      4981 non-null   int64  
 4   ever_married       4981 non-null   object 
 5   work_type          4981 non-null   object 
 6   Residence_type     4981 non-null   object 
 7   avg_glucose_level  4981 non-null   float64
 8   bmi                4962 non-null   float64
 9   smoking_status     4981 non-null   object 
 10  stroke             4981 non-null   int64  
dtypes: float64(3), int64(3), object(5)
memory usage: 428.2+ KB
```

Correlation

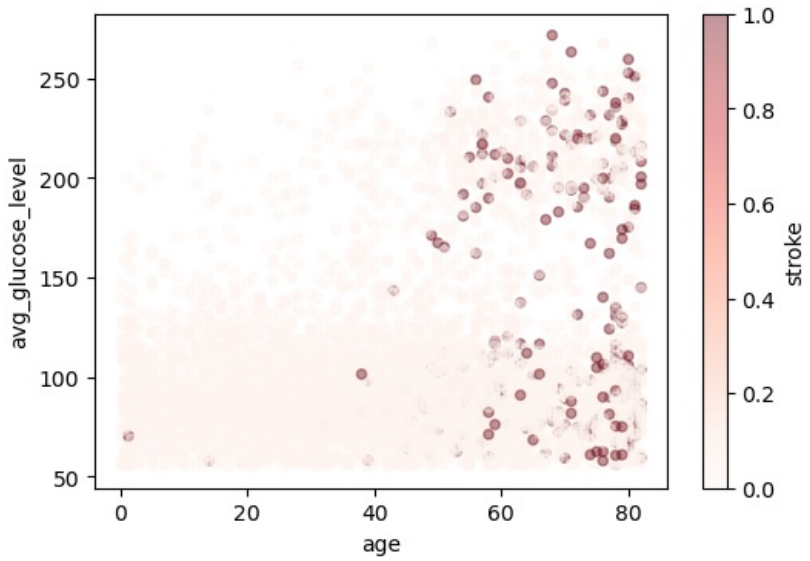
In [9]:
```
df = df.dropna()
df = df.drop_duplicates()
```

In [10]:
```
# View stroke Distribution
plt.figure(figsize=(15, 8))

# Define Color Palette
color_palette = {0:'blue', 1:'grey'}
distribution = df['stroke'].value_counts().sort_values(ascending=False)

bar_plot = sns.barplot(
    x=distribution.values,
    y=list(distribution.keys()),
    orient='horizontal', palette=color_palette
);

for container in bar_plot.containers:
    bar_plot.bar_label(container, fmt='%.0f', fontsize=11)

plt.title('Stroke Distribution', fontsize=20);
sns.despine();
plt.xlabel("Number of Samples", fontsize=16);
plt.ylabel("Stroke Label", fontsize=16);
```

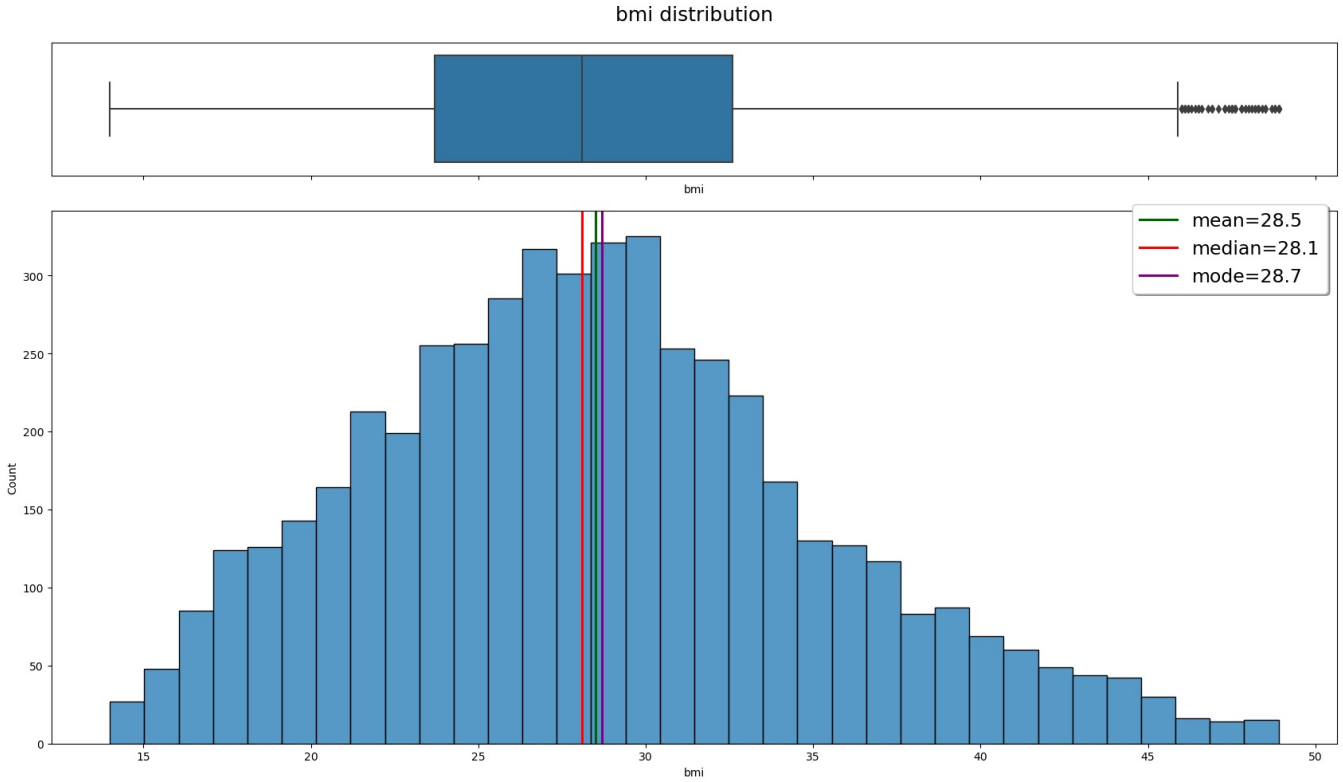


In [11]:
```
df.plot(kind="scatter", x="age", y="avg_glucose_level",
        c="stroke", cmap=plt.get_cmap("Reds"),
        colorbar=True, alpha=0.4, figsize=(6,4),
        )
```

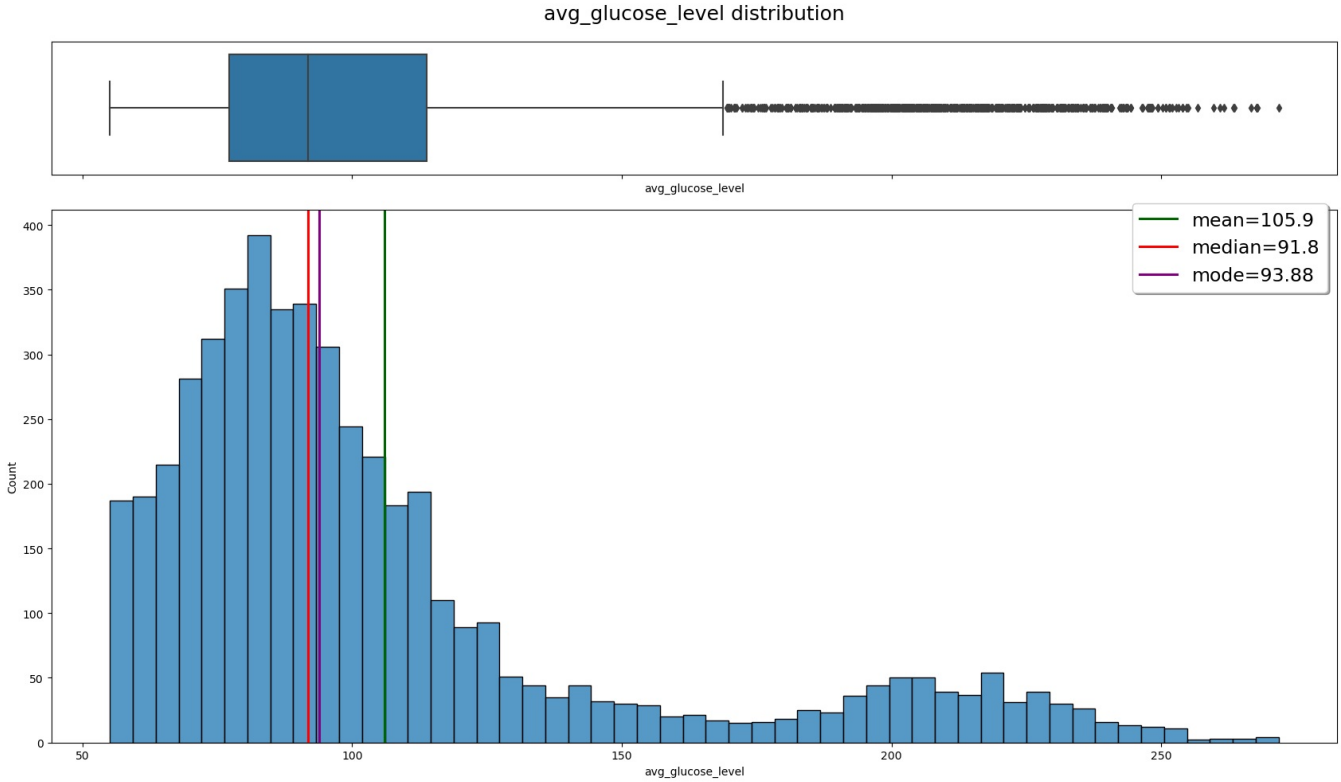
```
plt.show()
```



In [12]:
```
fig, ax = plt.subplots(2, 1, sharex=True, figsize=(17,10),gridspec_kw={"height_ratios": (.2, .8)})
ax[0].set_title('bmi distribution',fontsize=18,pad=20)
sns.boxplot(x='bmi', data=df, ax=ax[0])
ax[0].set(yticks=[])
sns.histplot(x='bmi', data=df, ax=ax[1])
plt.axvline(df['bmi'].mean(), color='darkgreen', linewidth=2.2, label='mean=' + str(np.round(df['bmi'].mean(),1
plt.axvline(df['bmi'].median(), color='red', linewidth=2.2, label='median='+ str(np.round(df['bmi'].median(),1)
plt.axvline(df['bmi'].mode()[0], color='purple', linewidth=2.2, label='mode='+ str(df['bmi'].mode()[0]))
plt.legend(bbox_to_anchor=(1, 1.03), ncol=1, fontsize=17, fancybox=True, shadow=True, frameon=True)
plt.tight_layout()
plt.show()
```

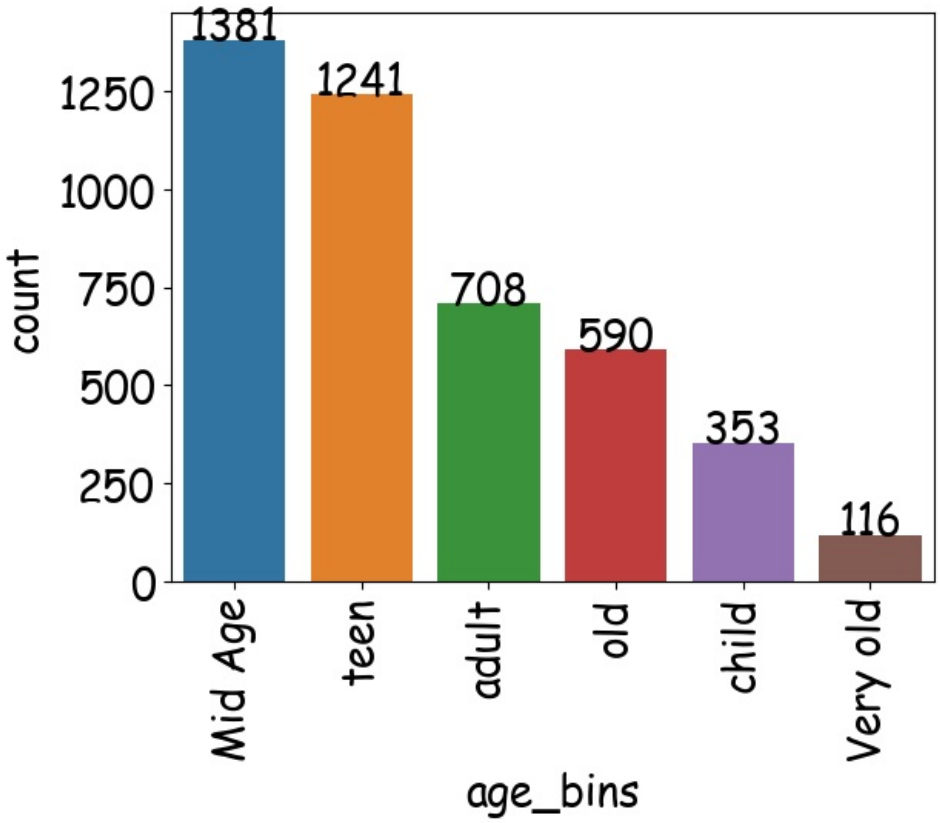


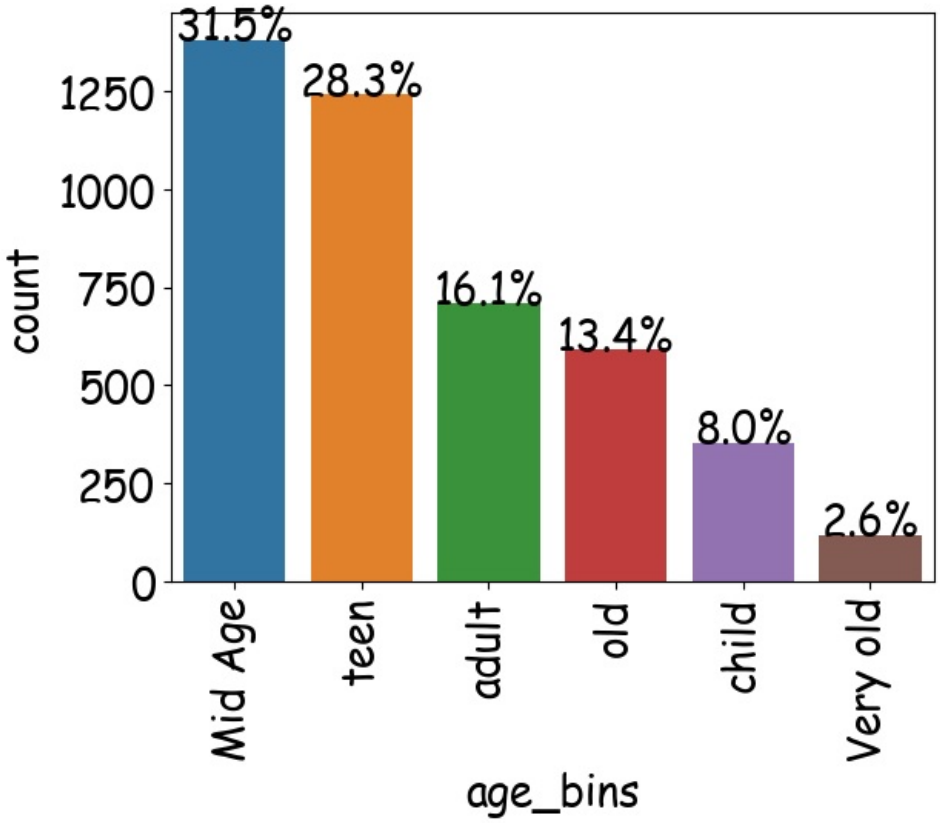
In [13]:
```
fig, ax = plt.subplots(2, 1, sharex=True, figsize=(17,10),gridspec_kw={"height_ratios": (.2, .8)})
ax[0].set_title('avg_glucose_level distribution',fontsize=18,pad=20)
sns.boxplot(x='avg_glucose_level', data=df, ax=ax[0])
ax[0].set(yticks=[])
sns.histplot(x='avg_glucose_level', data=df, ax=ax[1])
plt.axvline(df['avg_glucose_level'].mean(), color='darkgreen', linewidth=2.2, label='mean=' + str(np.round(df['
plt.axvline(df['avg_glucose_level'].median(), color='red', linewidth=2.2, label='median='+ str(np.round(df['avg
plt.axvline(df['avg_glucose_level'].mode()[0], color='purple', linewidth=2.2, label='mode='+ str(df['avg_glucos
plt.legend(bbox_to_anchor=(1, 1.03), ncol=1, fontsize=17, fancybox=True, shadow=True, frameon=True)
plt.tight_layout()
plt.show()
```



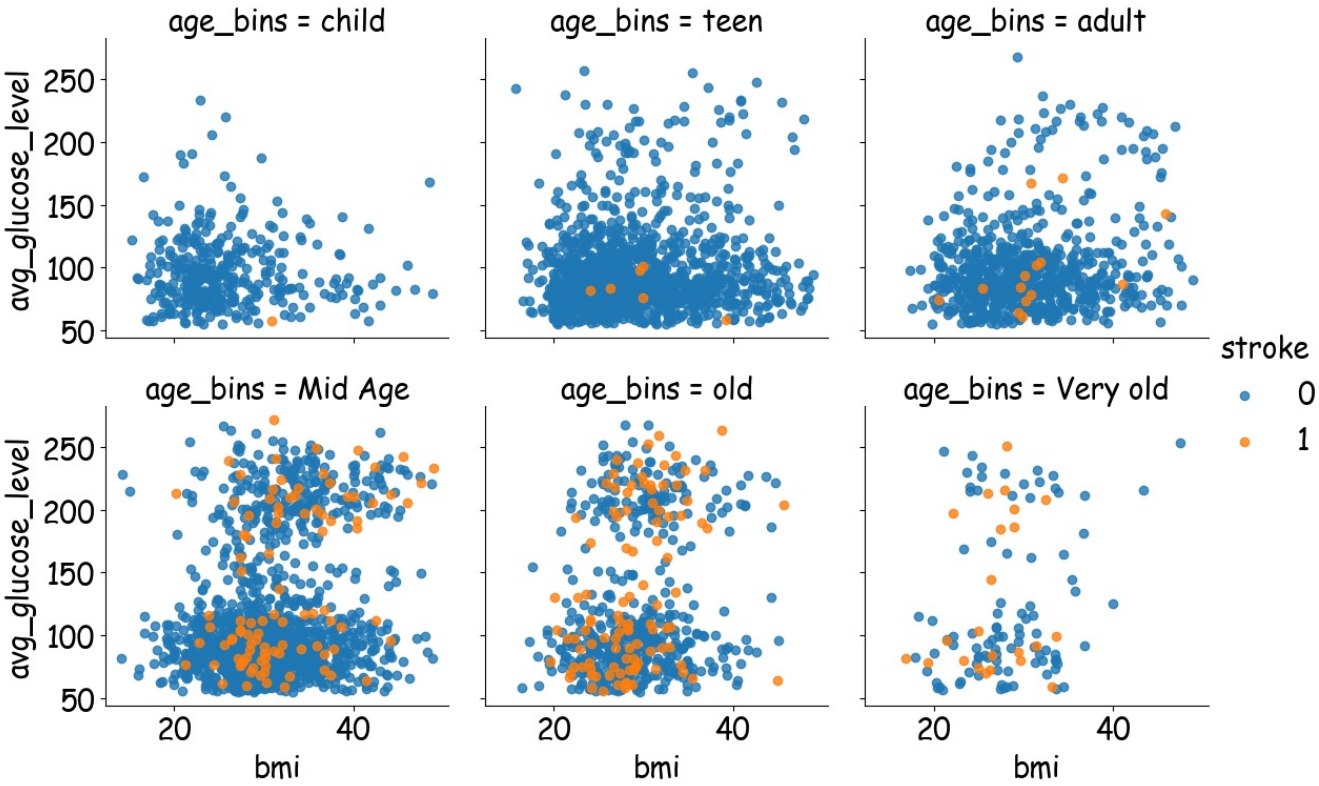
In [14]:
```
plt.rcParams['font.family'] = 'Comic Sans MS'
plt.rcParams['font.size'] = 20

df["age_bins"]= pd.cut(df["age"] , bins=[12, 19 , 40 , 50 , 70 ,  80,100] , labels=["child","teen", "adult" ,"M

def count_plot(data  , x=None , y=None  , figsize =None , title =None , color =None , prop=False , rotation_x =
    if x is None and y is None :
        raise("Expected y or x")
    if x is not None and y is not None:
        raise("Expected y or x not both")
    count_type = data[y if x is None else x].value_counts(ascending =False)
    Sum = count_type.sum()
    type_order = count_type.index
    plt.figure(figsize=figsize if figsize is None else (18 , 9))
    if x is None:
        sns.countplot(data = data , y=y , color = color ,order=type_order)
        if prop==True:
            for i in range(len(count_type)):
                count = count_type[i]
                pct_string ="{:0.1f}%".format(100*count/Sum)
                plt.text(count+1 , i , pct_string , va="center")
        if prop==False:
            for i in range(len(count_type)):
                count = count_type[i]
                pct_string ="{}".format(count)
                plt.text(count+1 , i , pct_string , va="center")
        plt.title(title)
        plt.show()
    if y is None :
        sns.countplot(data = data , x = x , color = color , order = type_order)
        locs , labels =plt.xticks(rotation = rotation_x)
        if prop == True :
            for loc , label in zip(locs , labels):
                count = count_type[label.get_text()]
                pct_string ="{:0.1f}%".format(100*count/Sum)
                plt.text(loc , count+2 ,pct_string,ha ="center")
        if prop==False :
            for loc , label in zip(locs , labels):
                count = count_type[label.get_text()]
                pct_string ="{}".format(count)
                plt.text(loc , count+2 ,pct_string,ha ="center")
        plt.title(title)
        plt.show()
```

In [15]:
```
count_plot(data=df,x='age_bins')

count_plot(data=df,x='age_bins',prop=True)
```

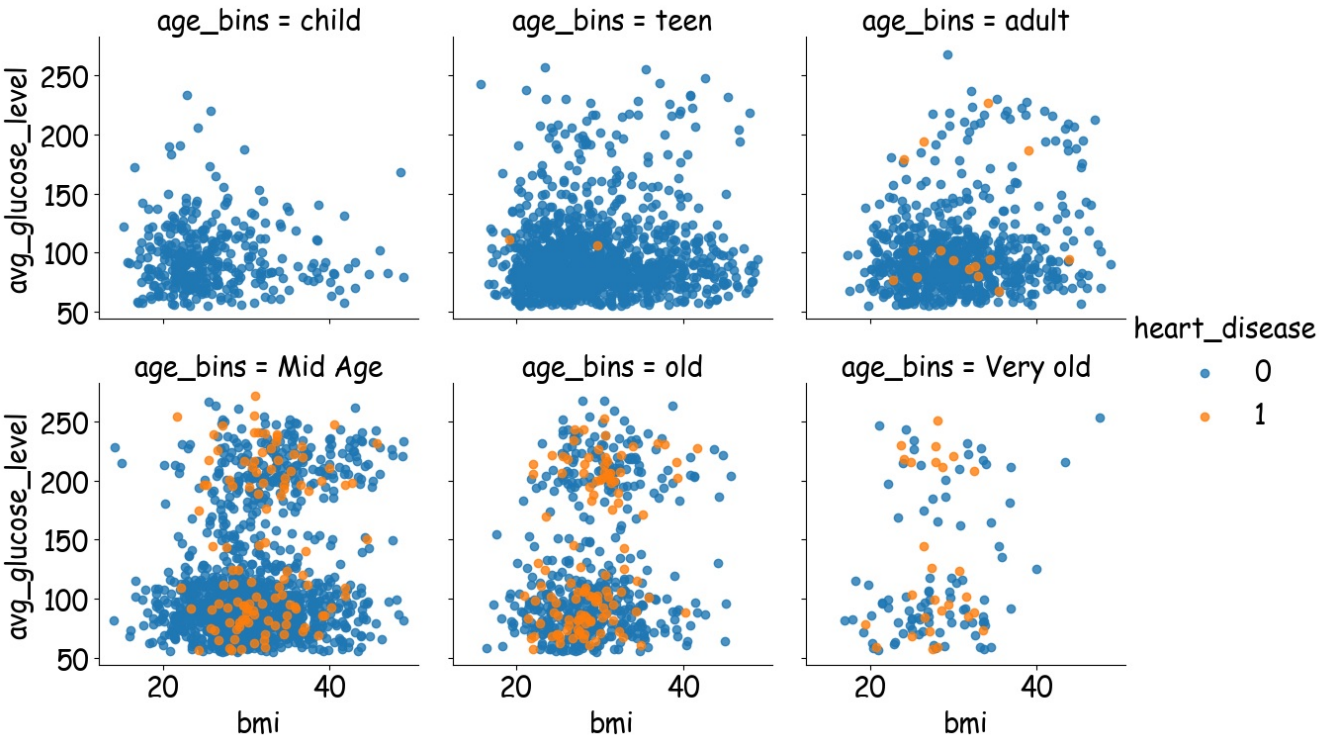




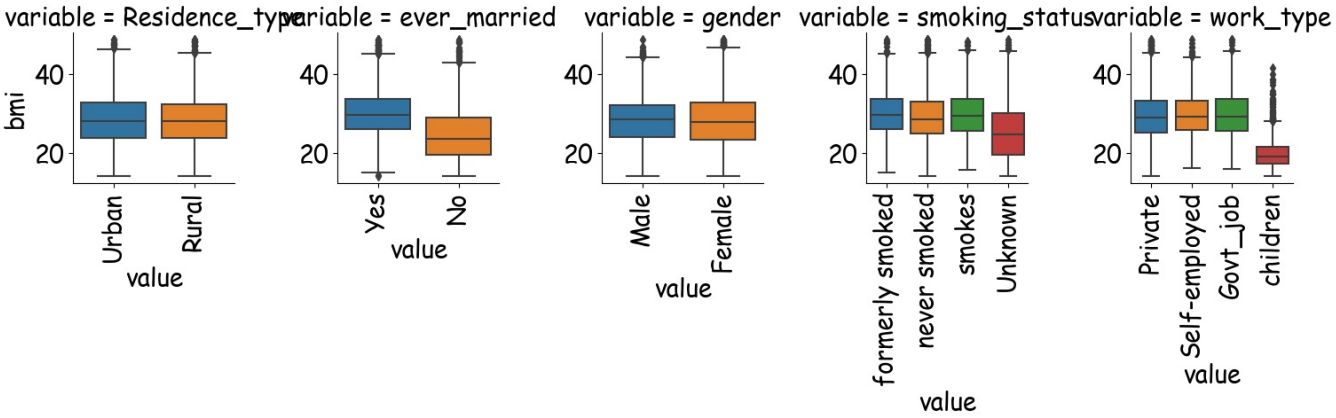
In [16]:
```
sns.lmplot(x="bmi",y="avg_glucose_level",hue="stroke",col="age_bins",height=4,col_wrap=3,data=df,fit_reg=False)
```



In [17]:
```
sns.lmplot(x="bmi",y="avg_glucose_level",hue="heart_disease",col="age_bins",height=4,col_wrap=3,data=df,fit_reg
```



In [18]:
```
categorical = df.select_dtypes(include=['object'])
##Visualising Categorical predictor variables with Target Variables
def facetgrid_boxplot(x, y, **kwargs):
    sns.boxplot(x=x, y=y)
    x=plt.xticks(rotation=90)

f = pd.melt(df, id_vars=['bmi'], value_vars=sorted(df[categorical.columns]))
g = sns.FacetGrid(f, col="variable", col_wrap=6, sharex=False, sharey=False)
g = g.map(facetgrid_boxplot, "value", "bmi")
```



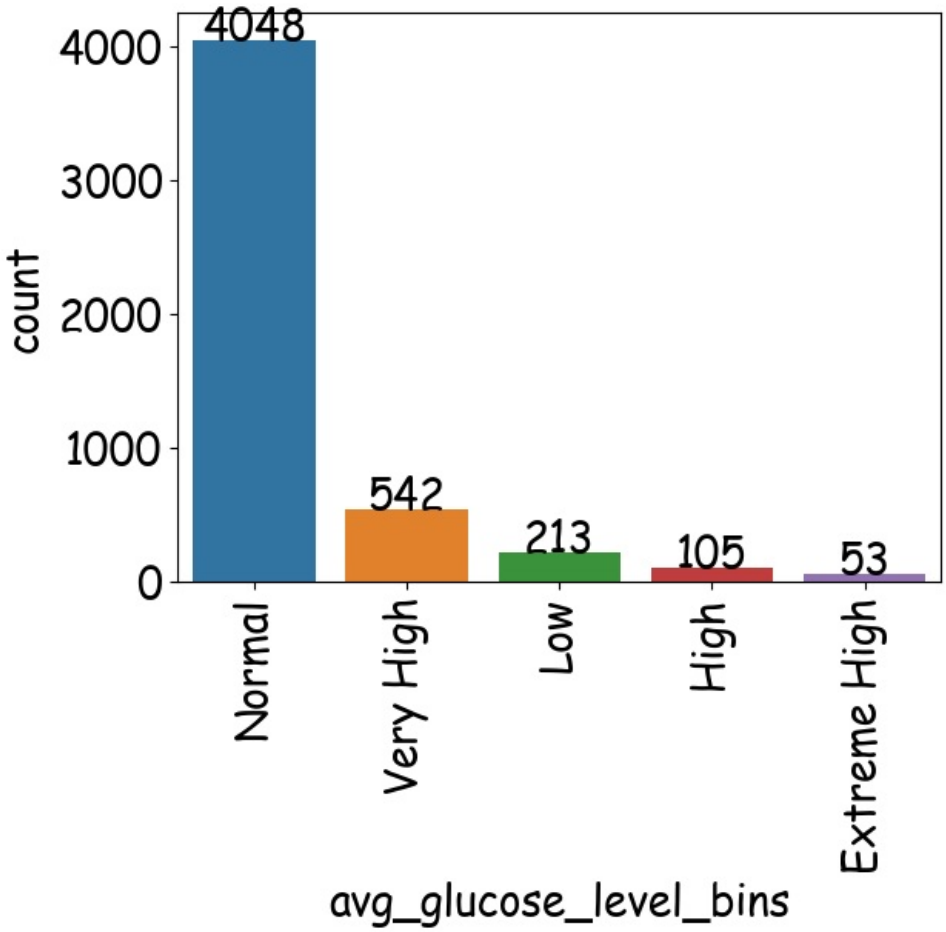
A1C %	Estimated Average Glucose (eAG)
10.0+	240+
9.5	226
9.0	212
8.5	197
8.0	183
7.6	171
7.5	169
7.3	163
7.1	157
7.0	154
6.7	146
6.5	140
<6.5	<140

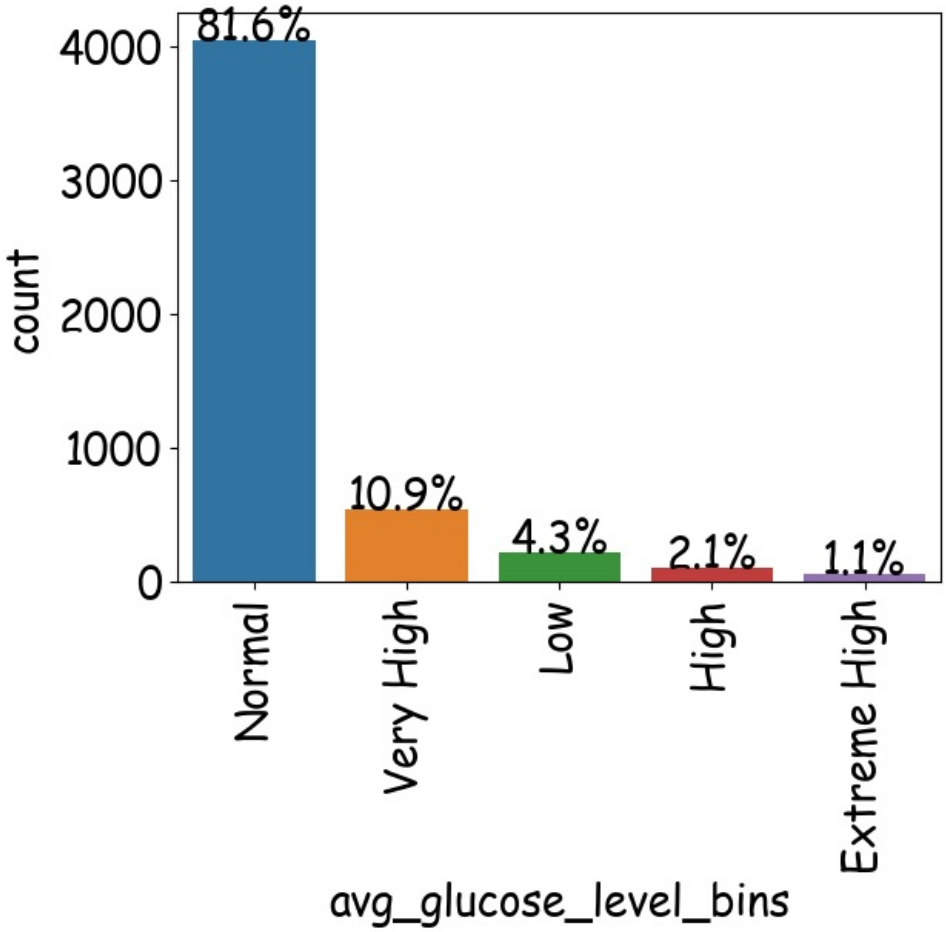
https://ihealthlabs.com/blogs/education/hemoglobin-a1c-and-estimated-average-glucose/

In [19]:
```
plt.rcParams['font.family'] = 'Comic Sans MS'
plt.rcParams['font.size'] = 20

df["avg_glucose_level_bins"]= pd.cut(df["avg_glucose_level"] , bins=[0, 60 , 150 , 170 , 240, 270 ] , labels=["

def count_plot(data  , x=None , y=None  , figsize =None , title =None , color =None , prop=False , rotation_x =
    if x is None and y is None :
        raise("Expected y or x")
    if x is not None and y is not None:
        raise("Expected y or x not both")
```

```
    count_type = data[y if x is None else x].value_counts(ascending =False)
    Sum = count_type.sum()
    type_order = count_type.index
    plt.figure(figsize=figsize if figsize is None else (18 , 9))
    if x is None:
        sns.countplot(data = data , y=y , color = color ,order=type_order)
        if prop==True:
            for i in range(len(count_type)):
                count = count_type[i]
                pct_string ="{:0.1f}%".format(100*count/Sum)
                plt.text(count+1 , i , pct_string , va="center")
        if prop==False:
            for i in range(len(count_type)):
                count = count_type[i]
                pct_string ="{}".format(count)
                plt.text(count+1 , i , pct_string , va="center")
        plt.title(title)
        plt.show()
    if y is None :
        sns.countplot(data = data , x = x , color = color , order = type_order)
        locs , labels =plt.xticks(rotation = rotation_x)
        if prop == True :
            for loc , label in zip(locs , labels):
                count = count_type[label.get_text()]
                pct_string ="{:0.1f}%".format(100*count/Sum)
                plt.text(loc , count+2 ,pct_string,ha ="center")
        if prop==False :
            for loc , label in zip(locs , labels):
                count = count_type[label.get_text()]
                pct_string ="{}".format(count)
                plt.text(loc , count+2 ,pct_string,ha ="center")
        plt.title(title)
        plt.show()
```

In [20]:
```
count_plot(data=df,x='avg_glucose_level_bins')

count_plot(data=df,x='avg_glucose_level_bins',prop=True)
```





In [21]:
```
df.head(1)
```

Out[21]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke	age
0	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1	M

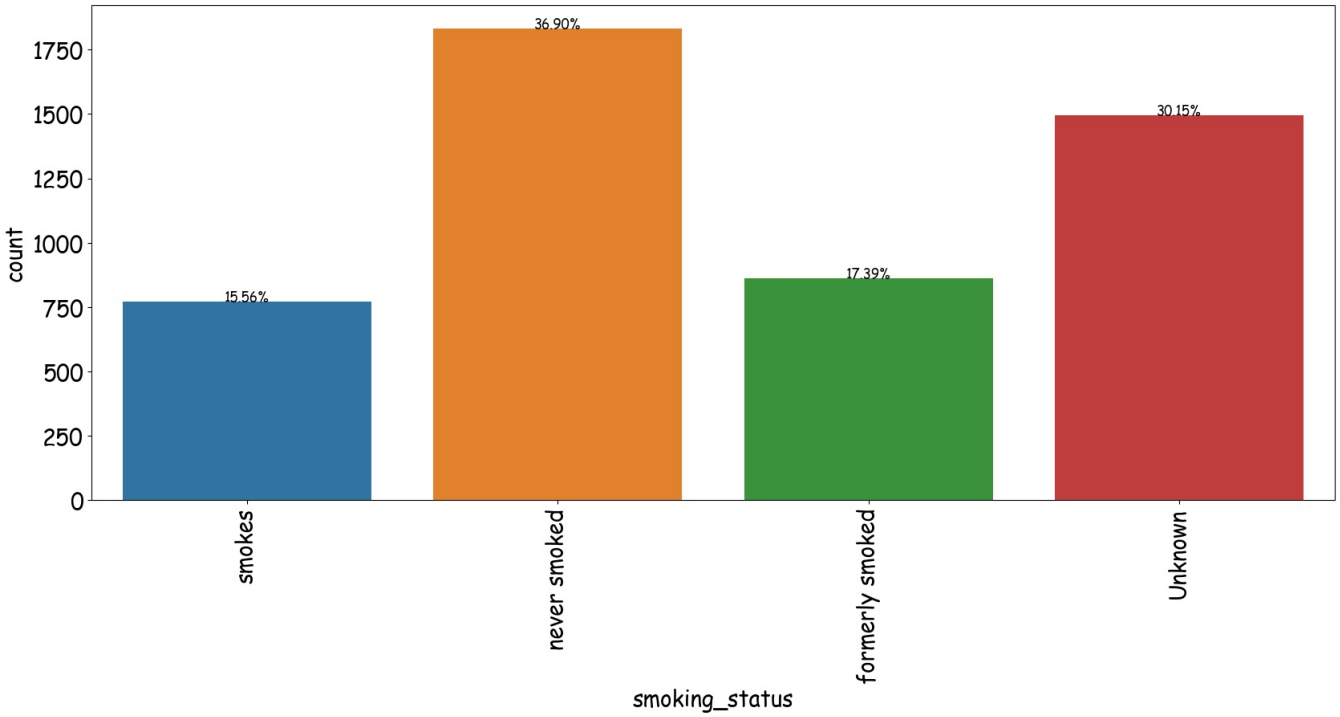
In [22]:
```
for col in df[['gender','hypertension','heart_disease','ever_married','work_type','Residence_type','smoking_sta
    print(f"\033[034m\033[1m",col,":","\n")
    print(f"\033[031m\033[1m",df[col].value_counts())
    print(f"\033[036m\033[1m","==="*15)
```

```
 gender :

 Female    2896
Male      2066
Name: gender, dtype: int64
 ==========================================
 hypertension :

 0    4485
1     477
Name: hypertension, dtype: int64
 ==========================================
 heart_disease :

 0    4688
1     274
Name: heart_disease, dtype: int64
 ==========================================
 ever_married :

 Yes    3266
No     1696
Name: ever_married, dtype: int64
 ==========================================
 work_type :

 Private          2851
Self-employed     801
children          671
Govt_job          639
Name: work_type, dtype: int64
 ==========================================
 Residence_type :

 Urban    2523
Rural    2439
Name: Residence_type, dtype: int64
 ==========================================
 smoking_status :

 never smoked       1831
Unknown            1496
formerly smoked     863
smokes              772
Name: smoking_status, dtype: int64
 ==========================================
 stroke :

 0    4715
1     247
Name: stroke, dtype: int64
 ==========================================
 age_bins :

 Mid Age     1381
teen        1241
adult        708
old          590
child        353
Very old     116
Name: age_bins, dtype: int64
 ==========================================
 avg_glucose_level_bins :

 Normal          4048
Very High        542
Low              213
High             105
Extreme High      53
Name: avg_glucose_level_bins, dtype: int64
 ==========================================
```

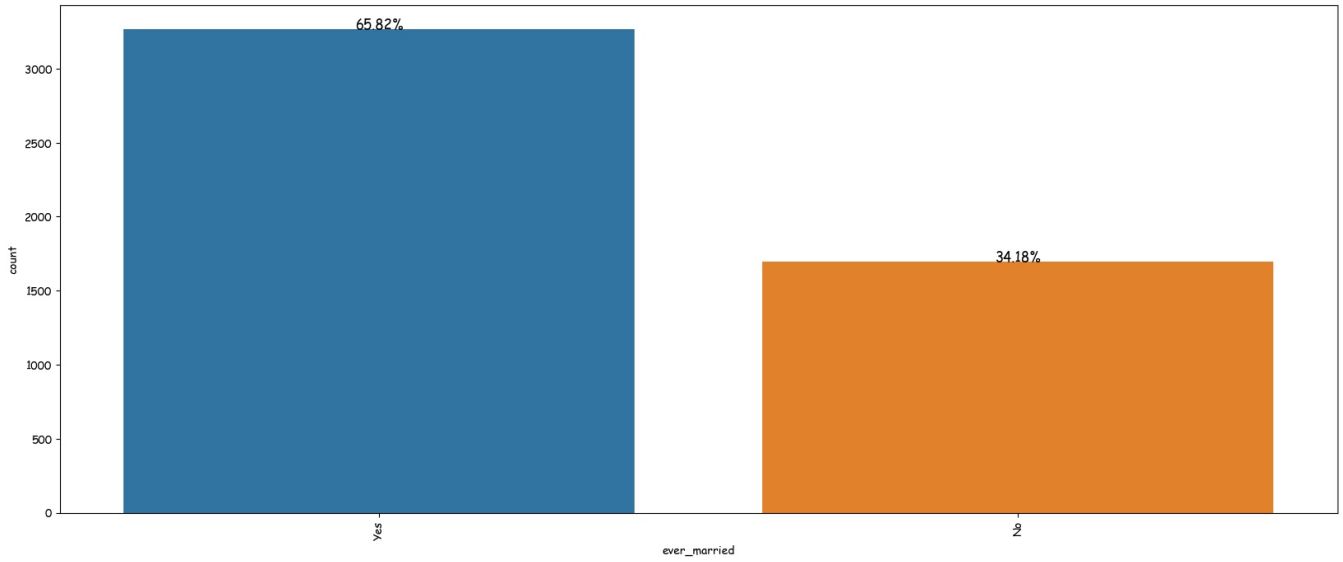
In [23]:

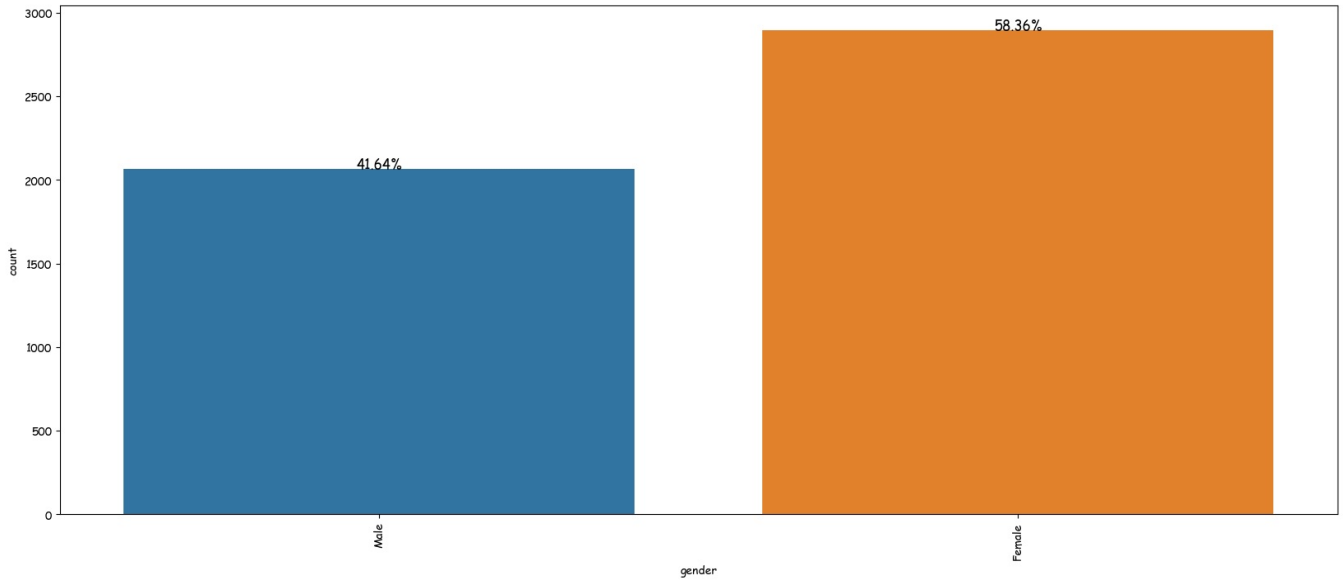
```
from wordcloud import WordCloud, STOPWORDS

text = " ".join(Company for Company in df["work_type"])
#font = "Quicksand-Bold.ttf"
word_cloud = WordCloud(width = 2300,
                       height = 800,
                       colormap = 'jet',
                       background_color = "white").generate(text)
plt.figure(figsize = (50, 8))
plt.imshow(word_cloud, interpolation = "gaussian")
plt.axis("off")
plt.show()
```

In [24]:
```
plt.figure(figsize = (20, 8))
s = sns.countplot(x = df['smoking_status'].sort_values(ascending=False),data = df)
sizes=[]
for p in s.patches:
    height = p.get_height()
    sizes.append(height)
    s.text(p.get_x()+p.get_width()/2.,
            height + 3,
            '{:1.2f}%'.format(height/len(df)*100),
            ha="center", fontsize=12,color = "Black")
    plt.xticks(rotation=90)
```

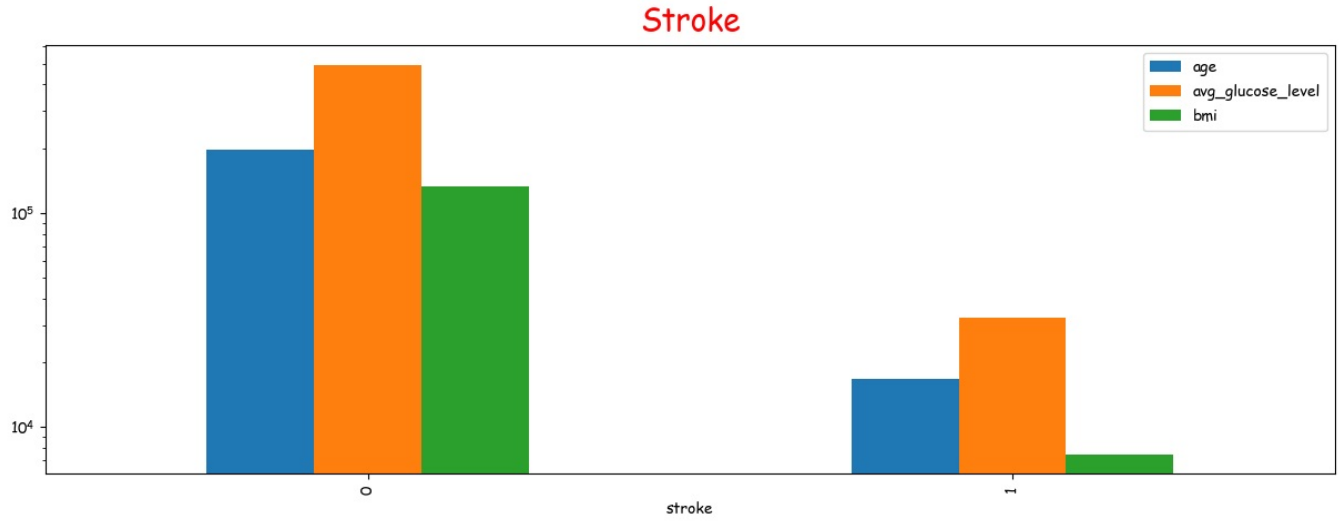


In [25]:
```
plt.rcParams['font.family'] = 'Comic Sans MS'
plt.rcParams['font.size'] = 10
plt.figure(figsize = (20, 8))
s = sns.countplot(x = df['ever_married'].sort_values(ascending=False),data = df)
sizes=[]
for p in s.patches:
    height = p.get_height()
    sizes.append(height)
    s.text(p.get_x()+p.get_width()/2.,
            height + 3,
            '{:1.2f}%'.format(height/len(df)*100),
            ha="center", fontsize=12,color = "Black")
    plt.xticks(rotation=90)

plt.figure(figsize = (20, 8))
s = sns.countplot(x = df['gender'].sort_values(ascending=False),data = df)
sizes=[]
for p in s.patches:
    height = p.get_height()
    sizes.append(height)
    s.text(p.get_x()+p.get_width()/2.,
            height + 3,
            '{:1.2f}%'.format(height/len(df)*100),
            ha="center", fontsize=12,color = "Black")
    plt.xticks(rotation=90)
```

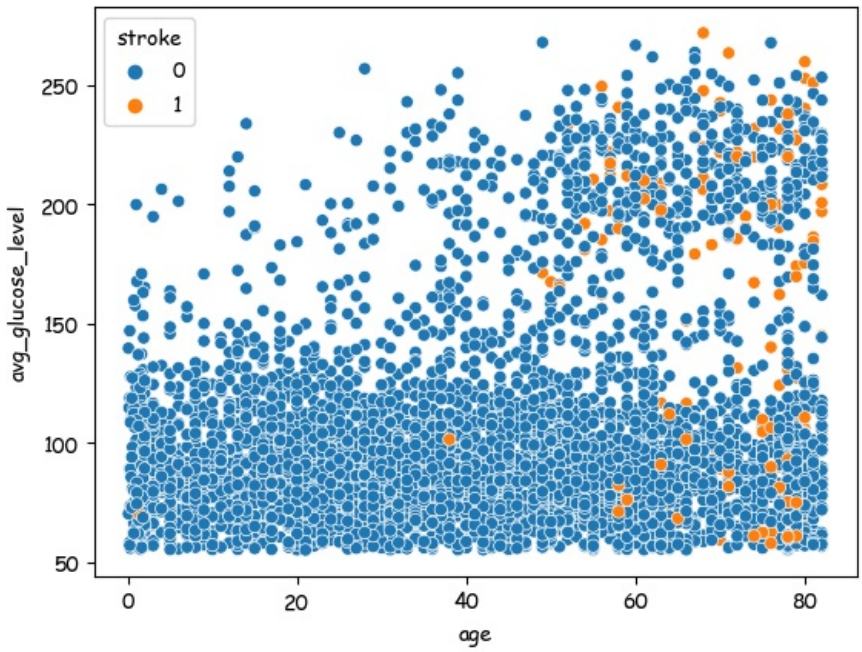




In [26]:
```
df.groupby('stroke')[['age', 'avg_glucose_level','bmi']].agg('sum').plot(rot=90, figsize=(15,5),kind='bar', log
plt.title('Stroke',color="Red",pad=10,fontsize=20)
plt.show()
```



In [27]:
```
sns.scatterplot(x="age", y="avg_glucose_level", hue="stroke", data=df,palette="tab10");
```



In [28]:
```
df.groupby(["gender",'ever_married','work_type'])['stroke'].value_counts().to_frame()
```

Out[28]:

				stroke
gender	**ever_married**	**work_type**	**stroke**	
Female	**No**	**Govt_job**	**0**	66
			1	3
		Private	**0**	497
			1	10
		Self-employed	**0**	58
			1	6
		children	**0**	314
			1	2
	Yes	**Govt_job**	**0**	297
			1	20
		Private	**0**	1129
			1	65
		Self-employed	**0**	395
			1	34
Male	**No**	**Govt_job**	**0**	42
			1	3
		Private	**0**	293
			1	4
		Self-employed	**0**	42
			1	1
		children	**0**	355
	Yes	**Govt_job**	**0**	201
			1	7
		Private	**0**	785
			1	68
		Self-employed	**0**	241
			1	24

In [29]:
```
stroke_Avg_Age = df.groupby("heart_disease")['age'].mean().to_frame().reset_index()
stroke_Avg_Age
```

Out[29]:

	heart_disease	age
0	0	41.976177
1	1	68.321168

In [30]:
```
stroke_Avg_Age = df.groupby("stroke")['age'].mean().to_frame().reset_index()
```

```
stroke_Avg_Age
```

Out[30]:

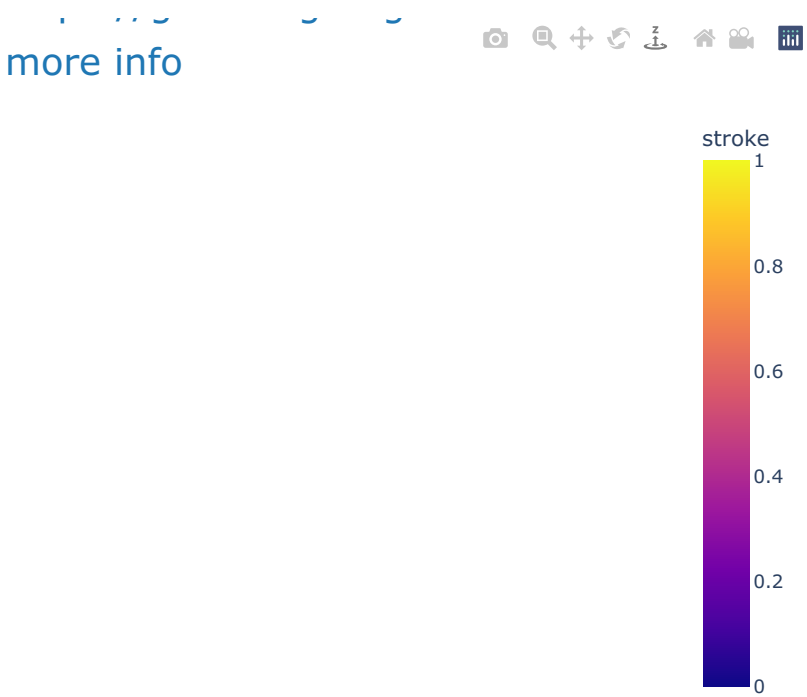
	stroke	age
0	0	42.155673
1	1	67.774575

In [31]:
```
WorkType_Avg_Age = df.groupby("work_type")['age'].mean().to_frame().reset_index()
WorkType_Avg_Age
```

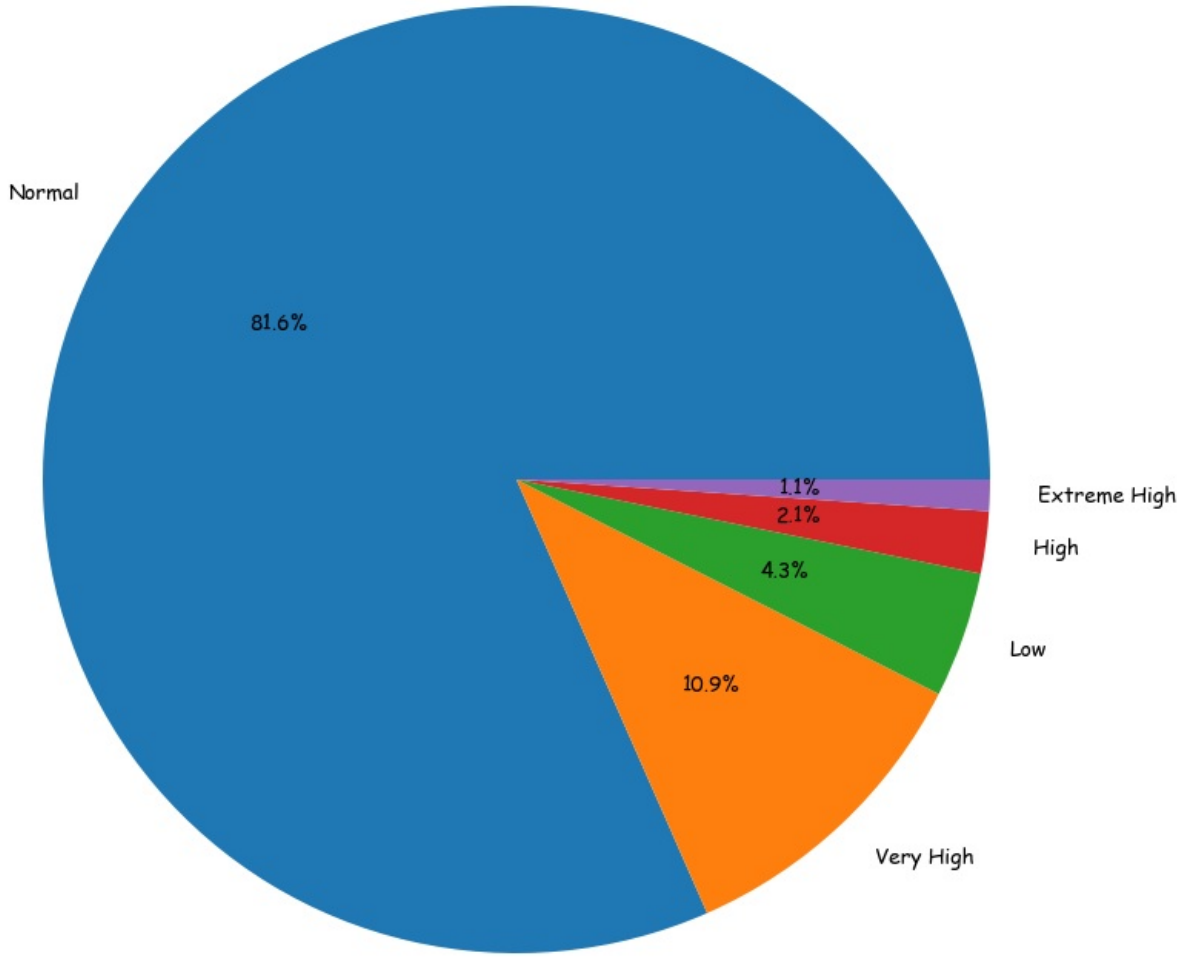
Out[31]:

	work_type	age
0	Govt_job	50.907668
1	Private	45.597334
2	Self-employed	60.360799
3	children	6.896155

In [32]:
```
# 3D Scatter Plot
import plotly.express as px
fig = px.scatter_3d(df, x='work_type', z='age', y='avg_glucose_level', color='stroke')
fig.show()
```



In [33]:
```
plt.figure(figsize=(20,10))
plt.pie(df['avg_glucose_level_bins'].value_counts(),labels=df["avg_glucose_level_bins"].value_counts().keys(),a
plt.show()
```



In [34]:
```
df.head()

pd.crosstab(df['work_type'],df['avg_glucose_level_bins'])[:10].style.background_gradient(cmap='copper')
```

Out[34]:

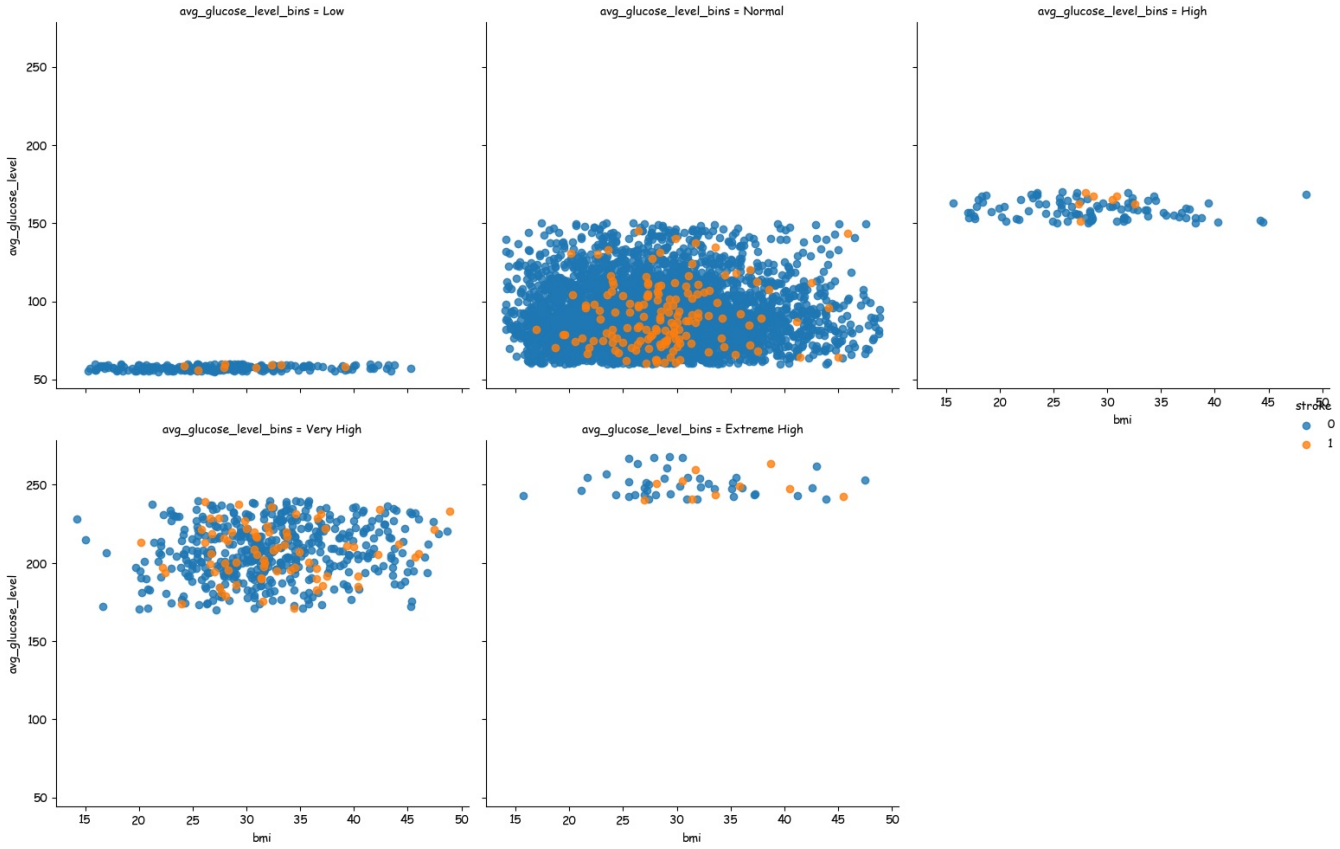
work_type	Low	Normal	High	Very High	Extreme High
Govt_job	32	504	17	79	7
Private	117	2318	58	326	31
Self-employed	31	612	18	125	15
children	33	614	12	12	0

In [35]:
```
pd.crosstab(df['work_type'],df['stroke'])[:10].style.background_gradient(cmap='copper')
```

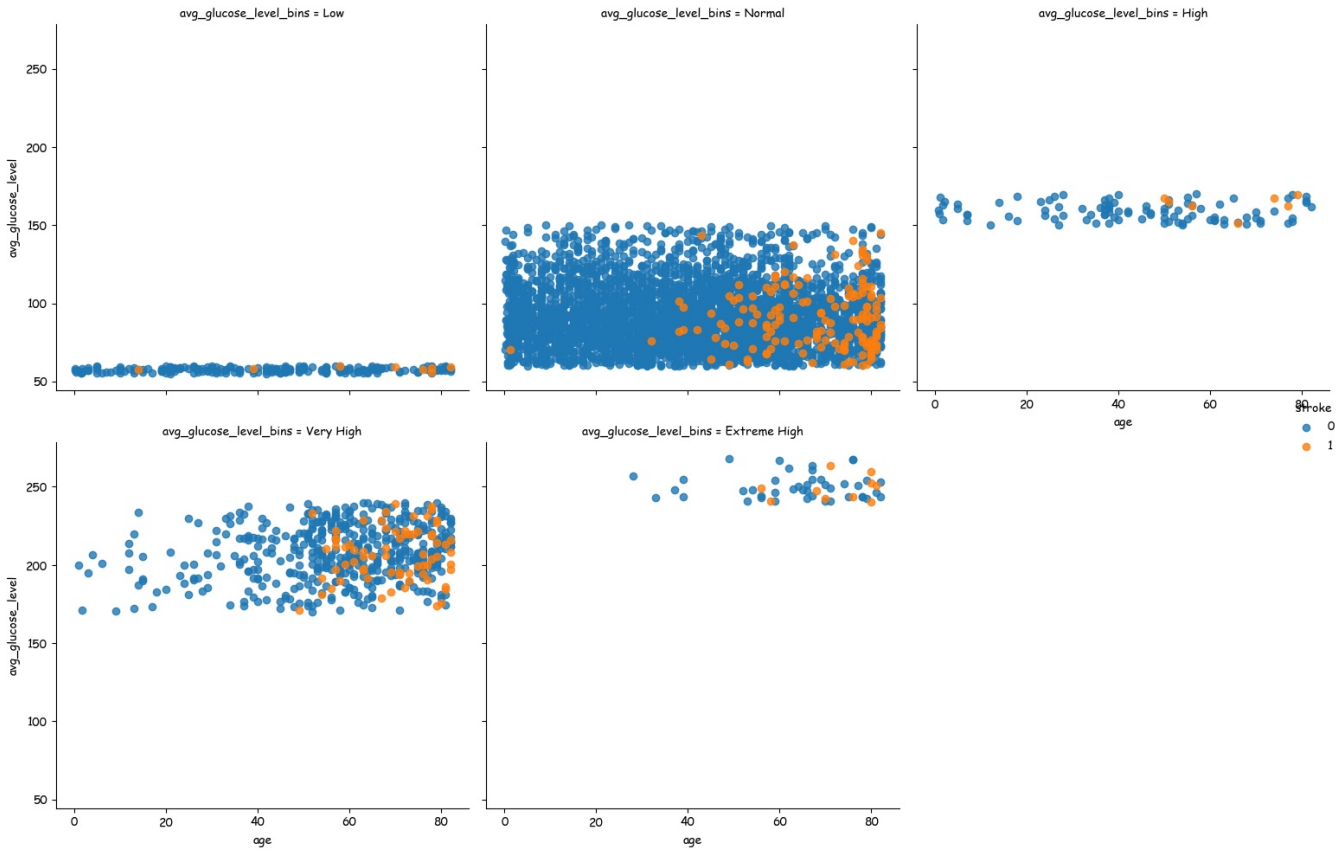
Out[35]:

work_type	0	1
Govt_job	606	33
Private	2704	147
Self-employed	736	65
children	669	2

In [36]:
```
sns.lmplot(x="bmi",y="avg_glucose_level",hue="stroke",col="avg_glucose_level_bins",height=5,col_wrap=3,data=df,
plt.tight_layout()
plt.show()
```

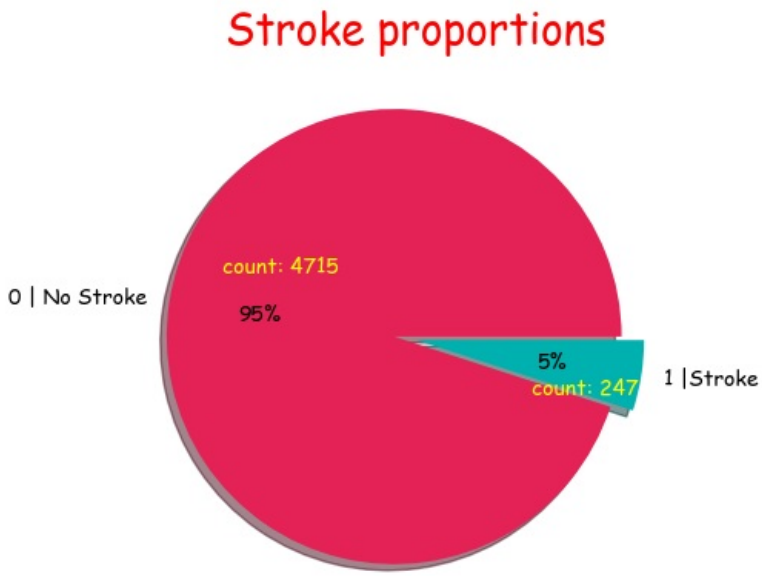


In [37]:
```
sns.lmplot(x="age",y="avg_glucose_level",hue="stroke",col="avg_glucose_level_bins",height=5,col_wrap=3,data=df,
plt.tight_layout()
plt.show()
```



In [38]:
```
plt.rcParams['font.family'] = 'Comic Sans MS'
plt.rcParams['font.size'] = 10


y_var = (
    df.loc[:, "stroke"]
    .value_counts()
    .reset_index()
    .rename(columns={"index": "stroke", "stroke": "counts"})
    .assign(percent=lambda df_: (df_["counts"] / df_["counts"].sum()).round(2) * 100)
)

target_color = sns.color_palette(["#e42256", "#00b1b0"])
```

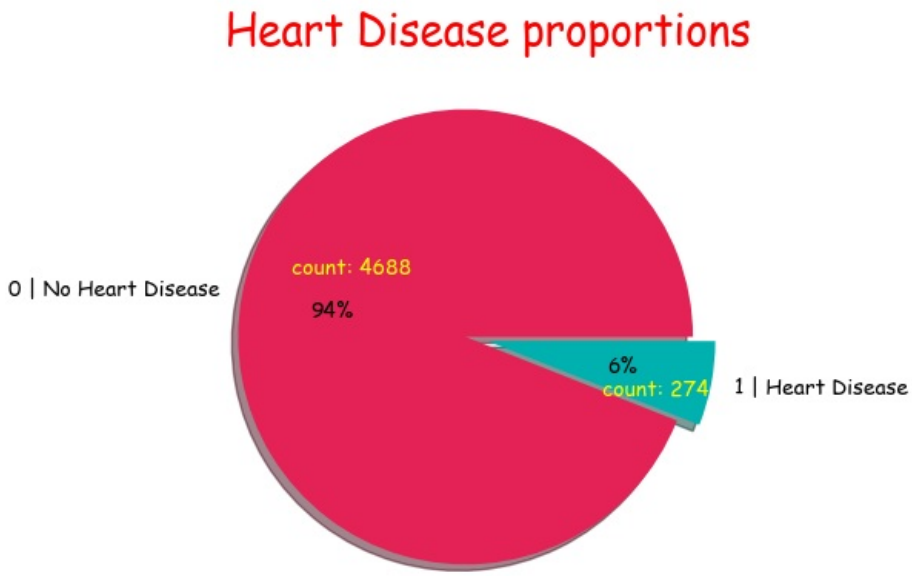
```
plt.pie(
    x=y_var["percent"],
    labels=["0 | No Stroke", "1 |Stroke"],
    colors=target_color,
    autopct="%.0f%%",
    shadow=True,
    textprops={"fontsize": 10, "color": "#000000"},
    explode=[0.1, 0],
)
plt.title("Stroke proportions", fontsize=20,color = "Red")
plt.text(
    x=-0.6,
    y=0.32,
    s="count: " + str(y_var.iloc[0, 1]) + "",
    color="Yellow",
    ha="center",
    va="center",
)
plt.text(
    x=0.74,
    y=-0.21,
    s="count: " + str(y_var.iloc[1, 1]) + "",
    color="Yellow",
    ha="center",
    va="center",
)
plt.show()
```



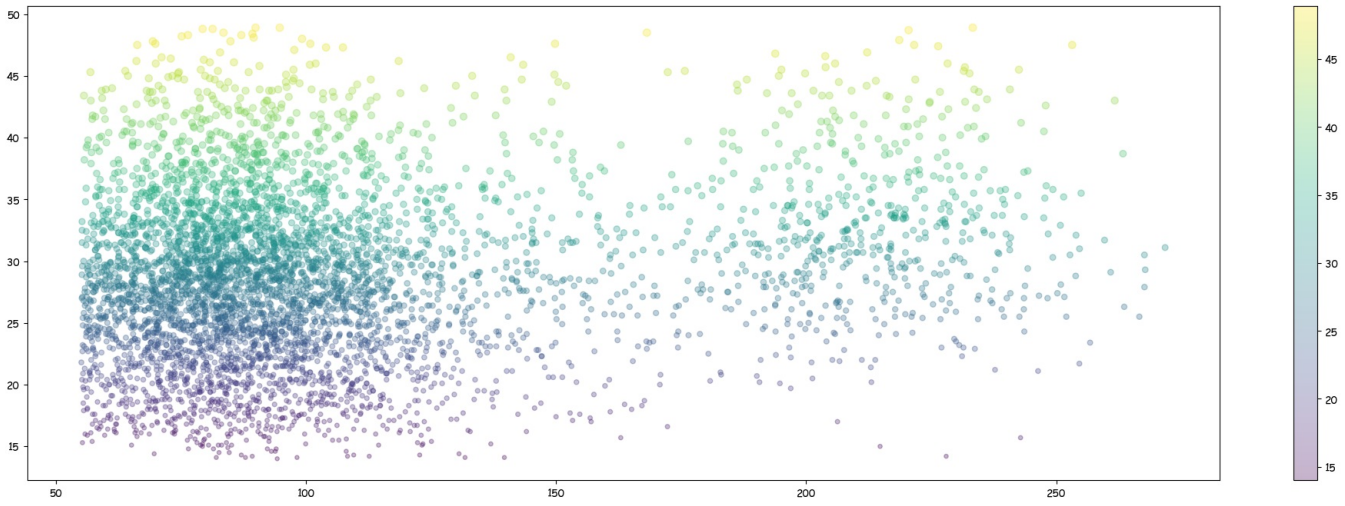
In [39]:

```
plt.rcParams['font.family'] = 'Comic Sans MS'
plt.rcParams['font.size'] = 10


y_var = (
    df.loc[:, "heart_disease"]
    .value_counts()
    .reset_index()
    .rename(columns={"index": "heart_disease", "heart_disease": "counts"})
    .assign(percent=lambda df_: (df_["counts"] / df_["counts"].sum()).round(2) * 100)
)

target_color = sns.color_palette(["#e42256", "#00b1b0"])

plt.pie(
    x=y_var["percent"],
    labels=["0 | No Heart Disease", "1 | Heart Disease"],
    colors=target_color,
    autopct="%.0f%%",
    shadow=True,
    textprops={"fontsize": 10, "color": "#000000"},
    explode=[0.1, 0],
)
plt.title("Heart Disease proportions", fontsize=20,color = "Red")
plt.text(
    x=-0.6,
    y=0.32,
    s="count: " + str(y_var.iloc[0, 1]) + "",
    color="Yellow",
    ha="center",
    va="center",
)
```

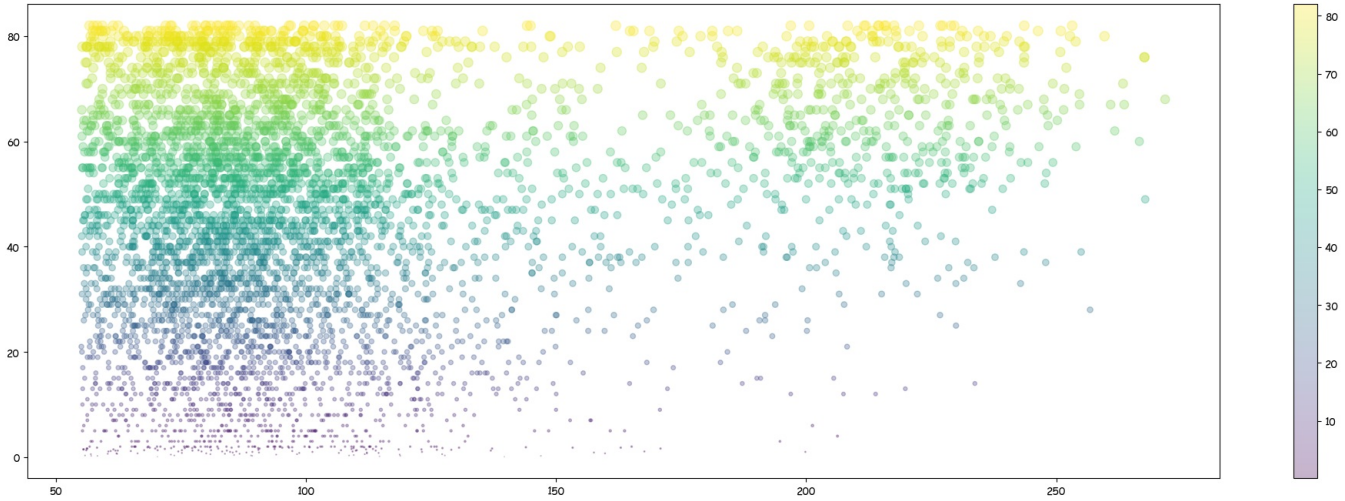
```
plt.text(
    x=0.74,
    y=-0.21,
    s="count: " + str(y_var.iloc[1, 1]) + "",
    color="Yellow",
    ha="center",
    va="center",
)
plt.show()
```



In [40]:
```
x = df["avg_glucose_level"]
y = df["bmi"]
colors = df["bmi"]
sizes = df["bmi"]

plt.figure(figsize = (25, 8))
plt.ticklabel_format(style = 'plain')
plt.scatter(x, y, c = colors, s = sizes, alpha = 0.3, cmap = 'viridis')
plt.colorbar();
```



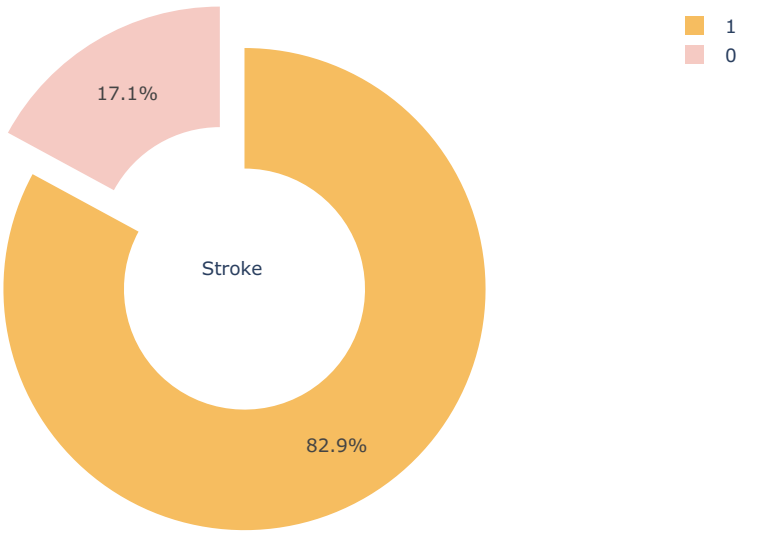
In [41]:
```
x = df["avg_glucose_level"]
y = df["age"]
colors = df["age"]
sizes = df["age"]

plt.figure(figsize = (25, 8))
plt.ticklabel_format(style = 'plain')
plt.scatter(x, y, c = colors, s = sizes, alpha = 0.3, cmap = 'viridis')
plt.colorbar();
```



In [42]:
```
df['avg_glucose_level_bins'].value_counts()
```

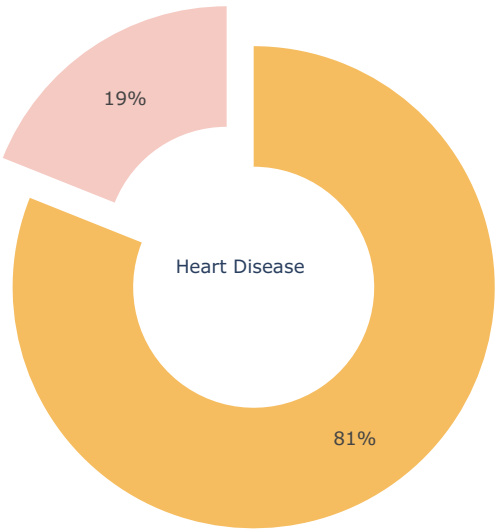
Out[42]:
```
Normal          4048
Very High        542
Low              213
High             105
Extreme High      53
Name: avg_glucose_level_bins, dtype: int64
```

In [43]:
```
import plotly.graph_objects as go

labels = HeartDisease['stroke'].unique()
values = HeartDisease['stroke'].value_counts()
palette = ["#f6bd60", "#f5cac3", "#84a59d", "#f28482"]

fig = go.Figure(data=[go.Pie(labels = labels,
                             values = values,
                             hole=.5,
                             title = 'Stroke',
                             legendgroup = True,
                             pull = [0.1, 0.1, 0.1, 0.1]
                            )
                      ]
                )

fig.update_traces(marker = dict(colors = palette))

fig.show()
```

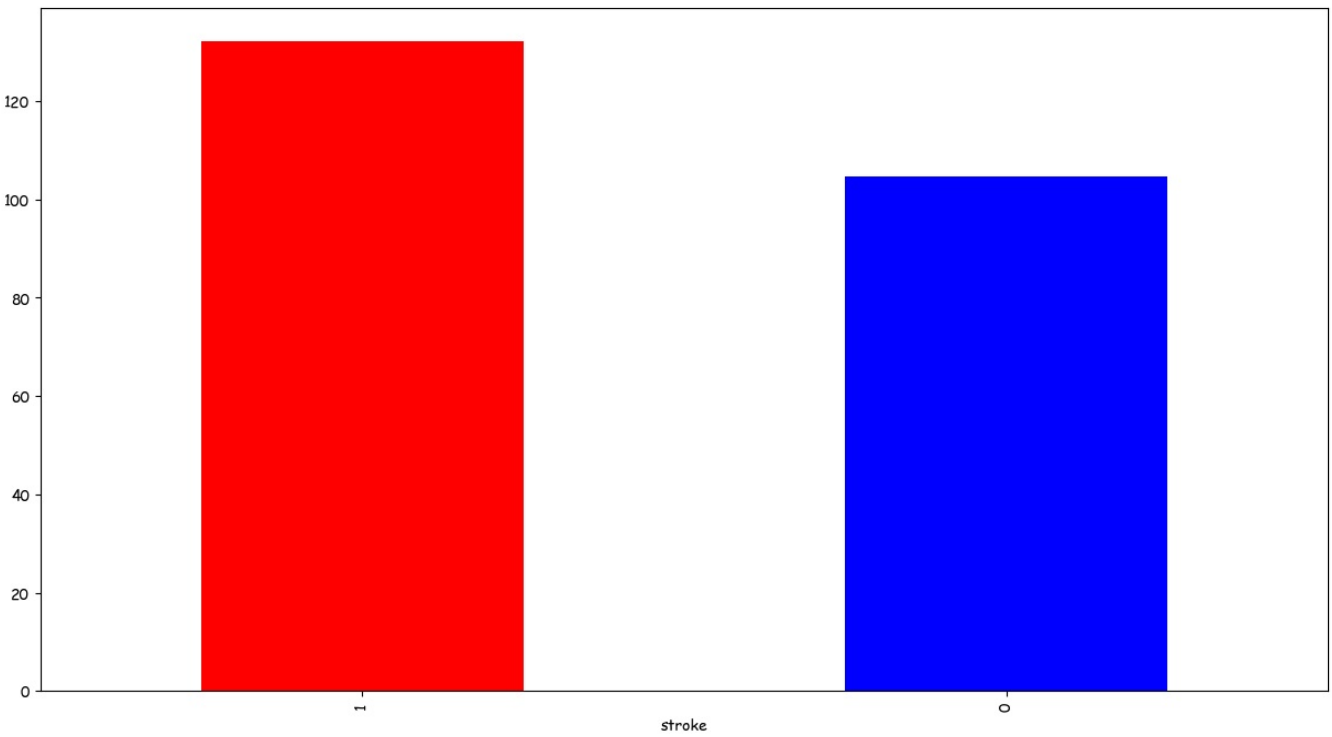


In [44]:
```
import plotly.graph_objects as go

labels = stroke['heart_disease'].unique()
values = stroke['heart_disease'].value_counts()
```

```
palette = ["#f6bd60", "#f5cac3"]

fig = go.Figure(data=[go.Pie(labels = labels,
                             values = values,
                             hole=.5,
                             title = 'Heart Disease',
                             legendgroup = True,
                             pull = [0.1, 0.1]
                            )
                      ]
                )

fig.update_traces(marker = dict(colors = palette))

fig.show()
```

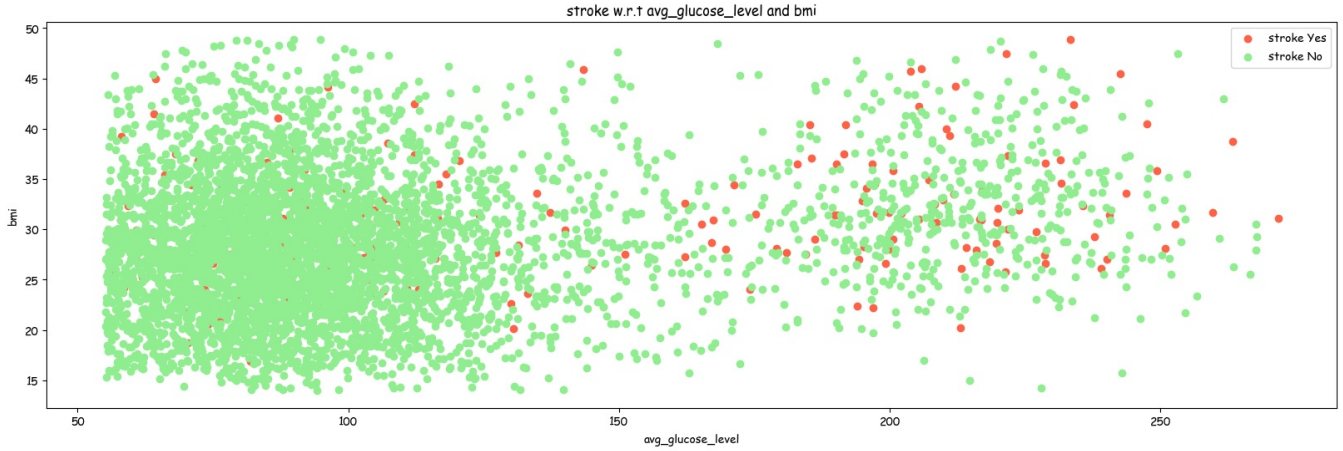


In [45]:
```
df2=df.groupby('stroke')['avg_glucose_level'].mean().sort_values(ascending=False)
plt.figure(figsize = (15,8))
color = [('b' if i < 130 else 'r') for i in df2]
df2.plot.bar(color=color);
```

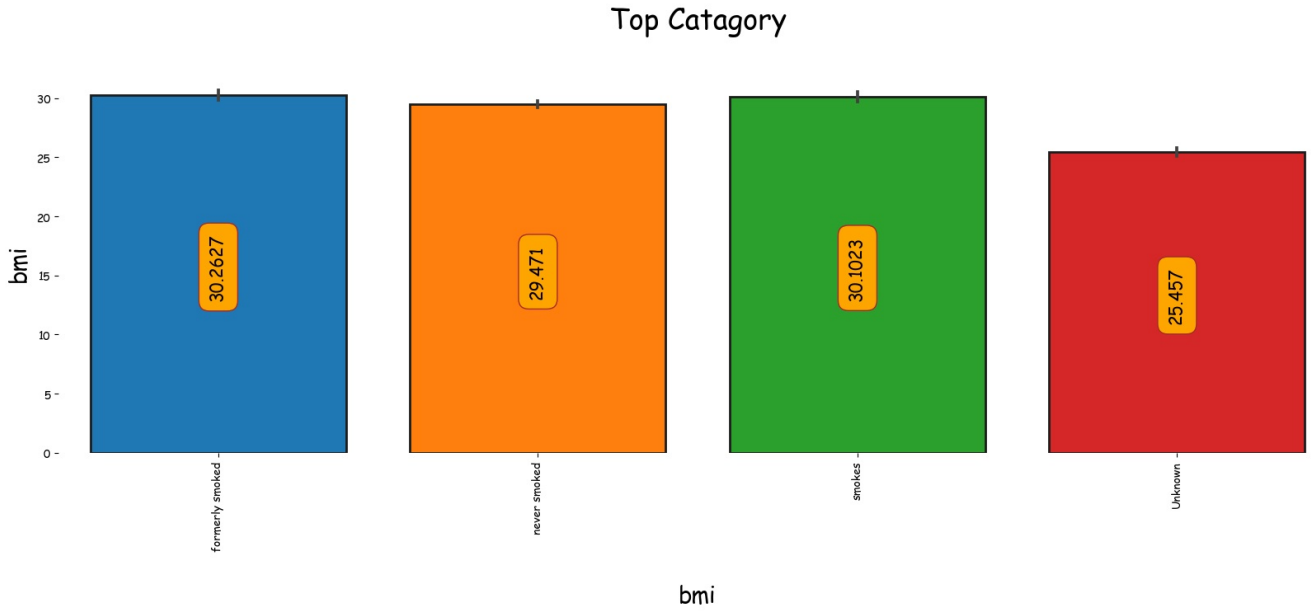


In [46]:
```
plt.figure(figsize=(20,6))
```

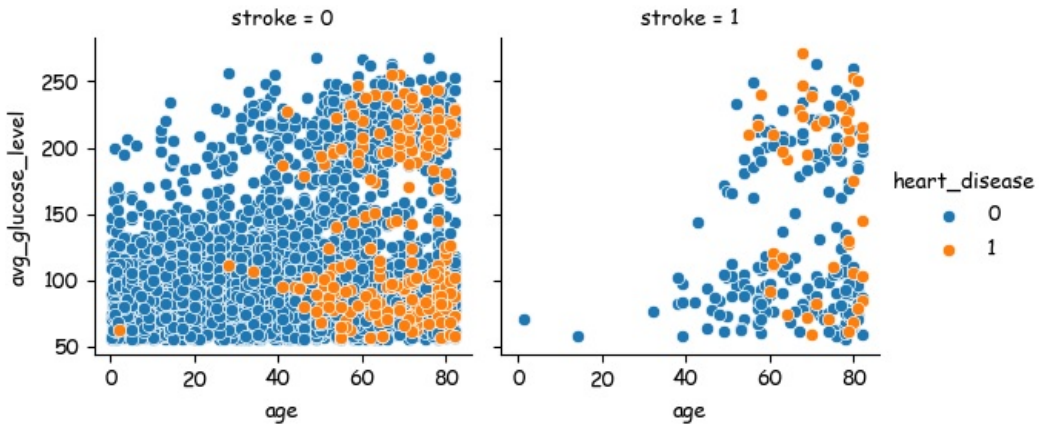
```
plt.scatter(df['avg_glucose_level'][df['stroke']==1],
            df['bmi'][df['stroke']==1],
            c="tomato")

plt.scatter(df['avg_glucose_level'][df['stroke']==0],
            df['bmi'][df['stroke']==0],
            c="lightgreen")

plt.title("stroke w.r.t avg_glucose_level and bmi")
plt.xlabel("avg_glucose_level")
plt.legend(["stroke Yes", "stroke No"])
plt.ylabel("bmi");
```



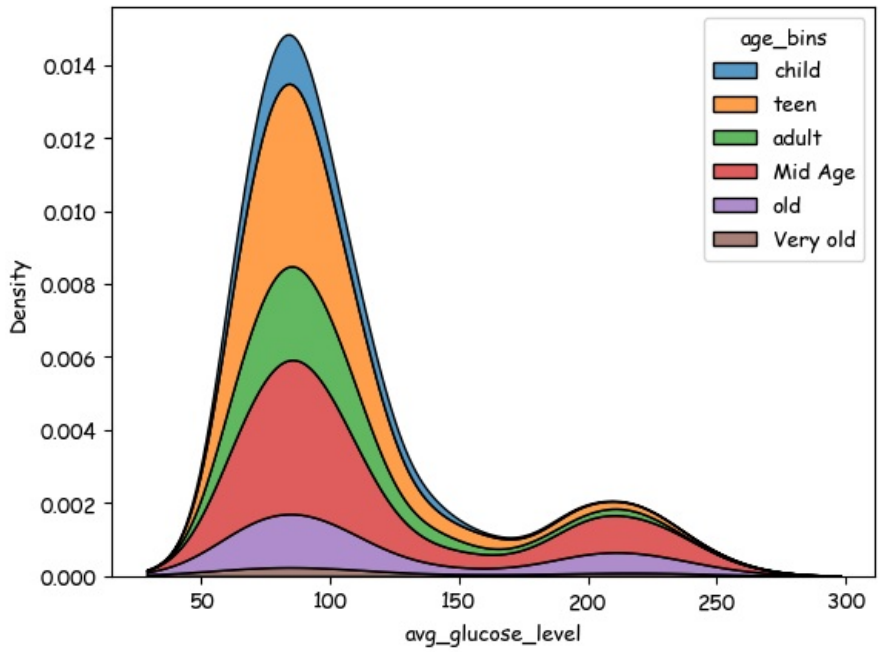
In [47]:
```
plt.subplots(figsize=(20,6))
p = sns.barplot(x=df["smoking_status"],y=df["bmi"],palette='tab10', saturation=1, edgecolor = "#1c1c1c", linewi
p.axes.set_title("\nTop Catagory\n", fontsize=25)
plt.ylabel("bmi" , fontsize = 20)
plt.xlabel("\nbmi" , fontsize = 20)
# plt.yscale("log")
plt.xticks(rotation = 90)
for container in p.containers:
    p.bar_label(container,label_type = "center",padding = 6,size = 15,color = "black",rotation = 90,
    bbox={"boxstyle": "round", "pad": 0.6, "facecolor": "orange", "edgecolor": "Brown", "alpha": 1})

sns.despine(left=True, bottom=True)
plt.show()
```



In [48]:
```
g = sns.FacetGrid(df, col="stroke", hue="heart_disease")
g.map_dataframe(sns.scatterplot, x="age", y="avg_glucose_level")
g.add_legend()
plt.show()
```

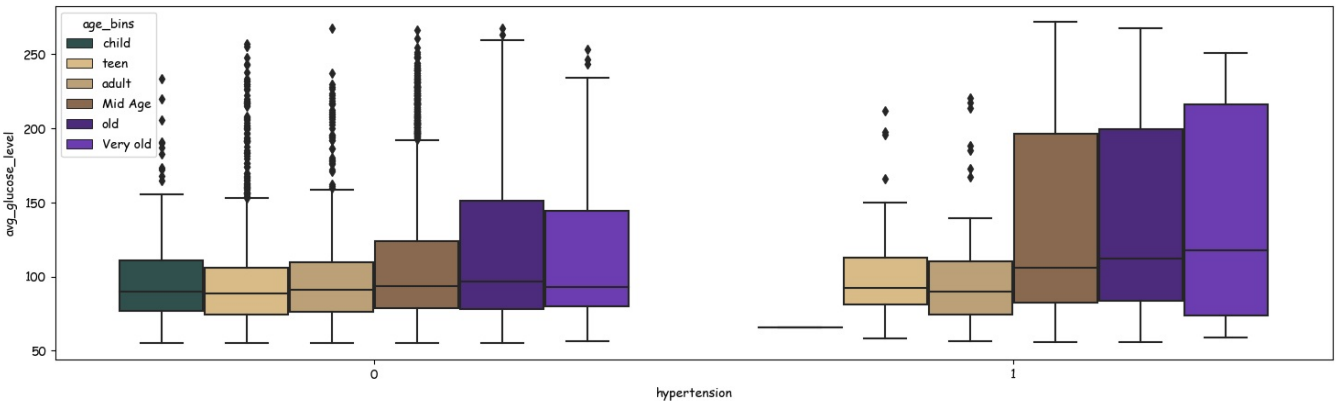


In [49]:
```
sns.kdeplot(data=df, x='avg_glucose_level', hue='age_bins', multiple='stack',palette='tab10');
```

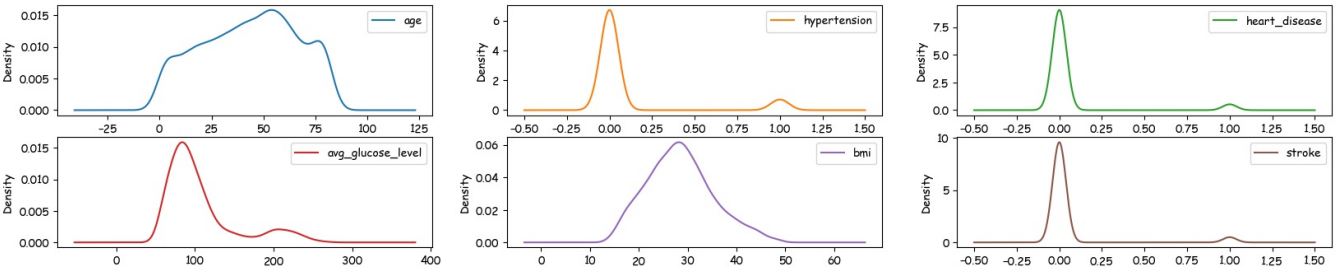


In [50]:
```
my_colors = ["#1a4441", "#2b5553", "#e6bf7b", "#c8a36c", "#936747",'#491D8B', '#6929C4', '#8A3FFC', '#A56EFF',
color_mix = ['#29066B', '#7D3AC1', '#AF4BCE', '#DB4CB2', '#EB548C']
black_grad = ['#100C07', '#3E3B39', '#6D6A6A', '#9B9A9C', '#CAC9CD']
```

In [51]:
```
plt.figure(figsize=(18,5))
sns.boxplot(x='hypertension', y='avg_glucose_level', hue='age_bins', data=df, linewidth=1.5, palette=my_colors[
```

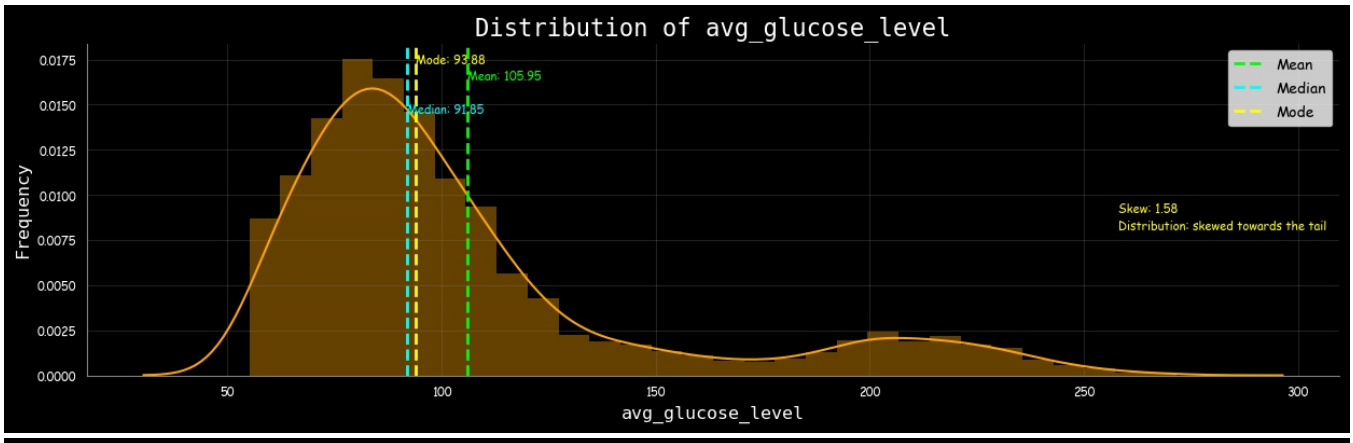


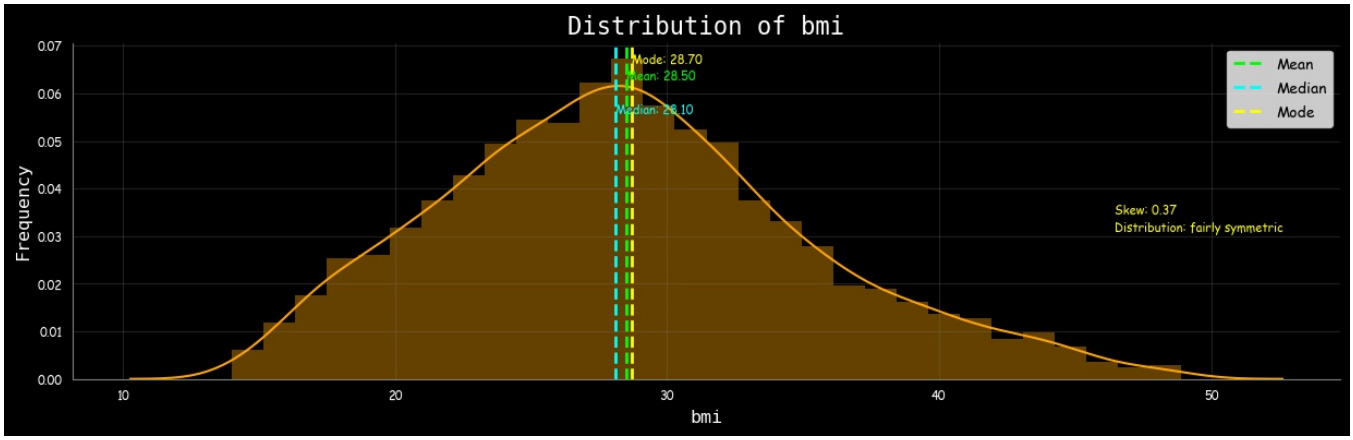
In [52]:
```
df.plot(kind='density', subplots=True, layout=(4,3), sharex=False, figsize= (20,8))
plt.show()
```



In [53]:

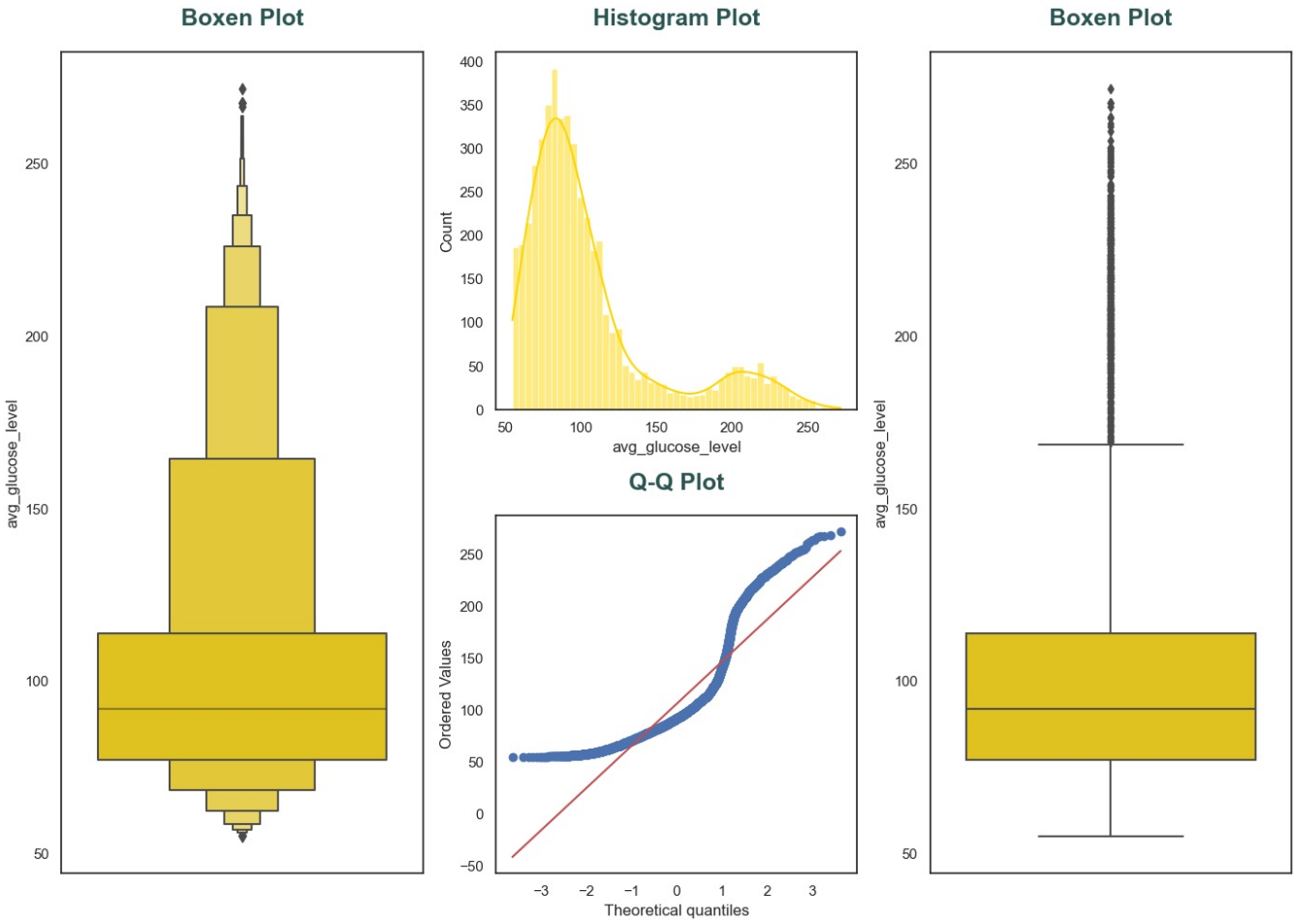
```
def skew_cat(skew_val):
    if skew_val>0.5:
        return 'skewed towards the tail'
    elif skew_val<-.5:
        return 'skewed towards the head'
    else:
        return 'fairly symmetric'

def skew_chart(x,var):

    median = x[var].median()
    mean = x[var].mean()
    mode = x[var].mode()[0]
    max=x[var].max()

    # Plotting the distribution of age
    fig, ax = plt.subplots(figsize=(15, 4))
    sns.distplot(x[var].dropna(), kde=True, bins=30, color='orange')

    # Adding title and axis labels
    plt.title("Distribution of "+ str(var), color='white',fontsize=16,fontdict={'fontname': 'monospace'})
    plt.xlabel(var, color='white',fontsize=12,fontdict={'fontname': 'monospace'})
    plt.ylabel("Frequency", color='white',fontsize=12,fontdict={'fontname': 'monospace'})

    fig.set_facecolor('black')  # Changing the background color
    ax.set_facecolor('black')  # Changing the plot area background color
    ax.grid(color='gray', alpha=0.25)  # Changing the grid lines color

    plt.axvline(mean, color='lime', linestyle='--', linewidth=2, label='Mean')  # Adding a vertical line for me
    plt.text(mean, ax.get_ylim()[1]*0.9, f"Mean: {mean:.2f}", color='lime', fontsize=8, ha='left', va='center')
    plt.axvline(median, color='cyan', linestyle='--', linewidth=2, label='Median')  # Adding a vertical line fo
    plt.text(median, ax.get_ylim()[1]*0.8, f"Median: {median:.2f}", color='cyan', fontsize=8, ha='left', va='ce
    plt.axvline(mode, color='yellow', linestyle='--', linewidth=2, label='Mode')  # Adding a vertical line for
    plt.text(mode, ax.get_ylim()[1]*0.95, f"Mode: {mode:.2f}", color='yellow', fontsize=8, ha='left', va='cente
    plt.text(max*.95, ax.get_ylim()[1]*0.5, f"Skew: {x[var].skew():.2f}", color='yellow', fontsize=8, ha='left'
    plt.text(max*.95, ax.get_ylim()[1]*0.45, f"Distribution: {skew_cat(x[var].skew())}", color='yellow', fontsi
    plt.yticks(color='white',fontsize=8) # Changing the font size of y-axis ticks
    plt.xticks(color='white',fontsize=8) # Changing the font size of x-axis ticks

    plt.setp(ax.spines.values(), color='gray') # Changing the color of the spines
    sns.despine()  # Removing the top and right spines
    plt.legend()  # Adding the legend
    plt.show()  # Displaying the plot

#creating list by type of variables
continuous=['avg_glucose_level','bmi'] # creating a list of continuous attributes
categorical=['gender','ever_married','work_type','smoking_status'] # creating a list of categorical attributes
target=['stroke'] # creating a list of target attribute

[skew_chart(df,i) for i in continuous]
```



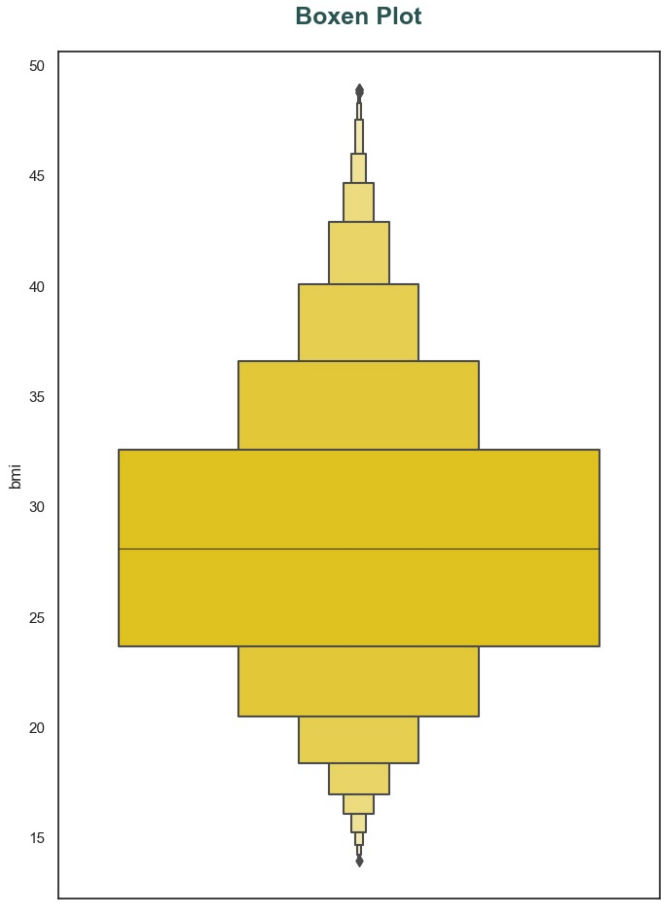


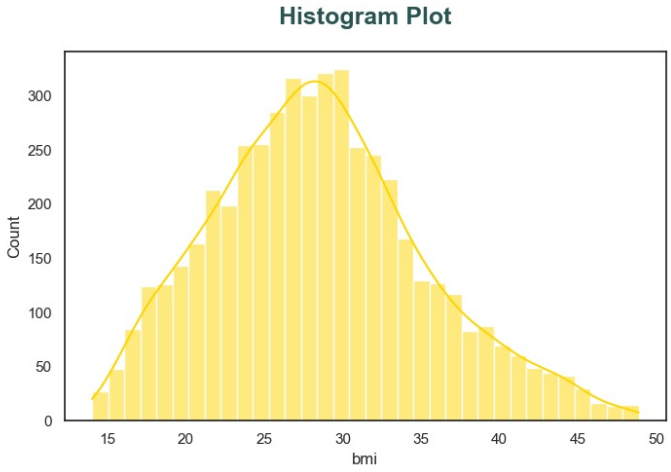
Out[53]: [None, None]

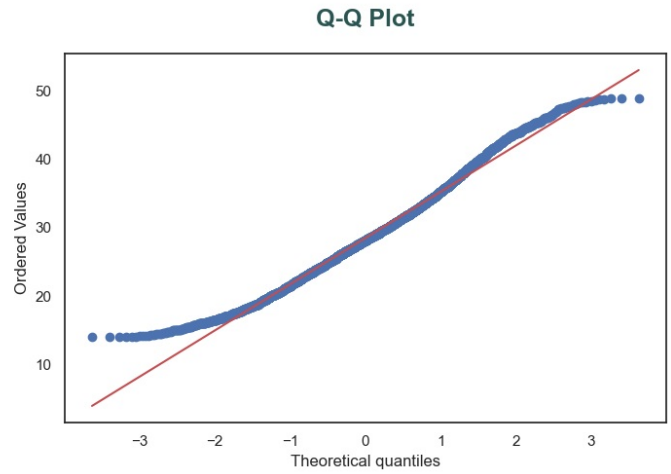
In [54]:
```
from scipy import stats
sns.set_theme(style="white")
from scipy.stats import norm
fig = plt.figure(figsize = (14, 10))
ax_1 = fig.add_subplot(2, 3, 2)
sns.histplot(data = df, x = 'avg_glucose_level', kde = True, color = "Gold")
plt.title('Histogram Plot', fontweight = 'bold', fontsize = 18, fontfamily = 'sans-serif', color = my_colors[1]
ax_2 = fig.add_subplot(2, 3, 5)
stats.probplot(df['avg_glucose_level'], plot=plt)
plt.title('Q-Q Plot', fontweight = 'bold', fontsize = 18, fontfamily = 'sans-serif', color = my_colors[1],pad=2
ax_3 = fig.add_subplot(1, 3, 1)
sns.boxenplot(y = 'avg_glucose_level', data = df, color = "Gold", linewidth = 1.5);
plt.title('Boxen Plot', fontweight = 'bold', fontsize = 18, fontfamily = 'sans-serif', color = my_colors[1],pad
ax_4 = fig.add_subplot(1, 3, 3)
sns.boxplot(y = 'avg_glucose_level', data = df, color = "Gold", linewidth = 1.5);
plt.title('Boxen Plot', fontweight = 'bold', fontsize = 18, fontfamily = 'sans-serif', color = my_colors[1],pad
plt.tight_layout();
plt.show();
```



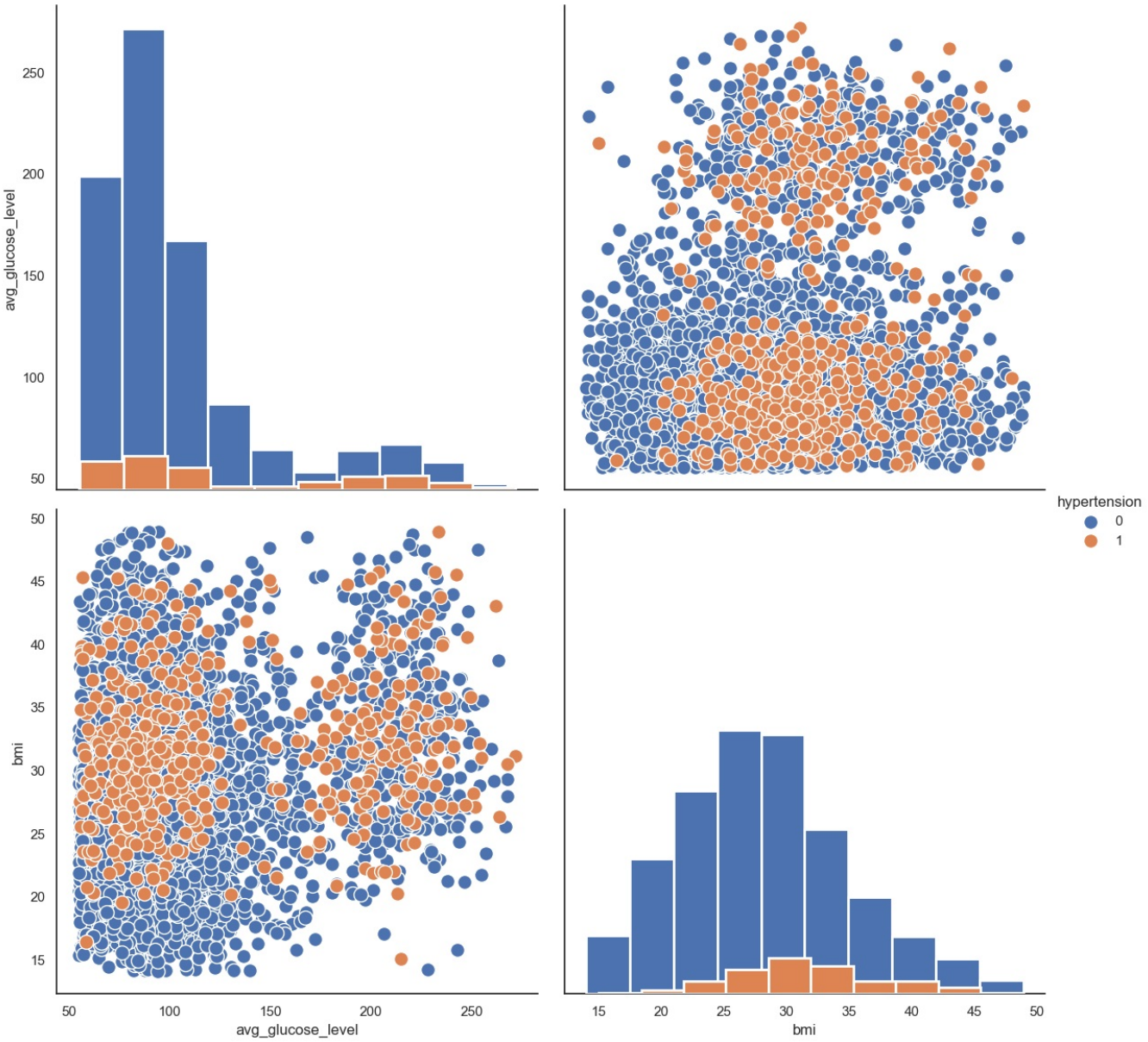
In [55]:
```
from scipy import stats
sns.set_theme(style="white")
from scipy.stats import norm
fig = plt.figure(figsize = (14, 10))
ax_1 = fig.add_subplot(2, 2, 2)
sns.histplot(data = df, x = 'bmi', kde = True, color = "Gold")
plt.title('Histogram Plot', fontweight = 'bold', fontsize = 18, fontfamily = 'sans-serif', color = my_colors[1]
ax_2 = fig.add_subplot(2, 2, 4)
stats.probplot(df['bmi'], plot=plt)
plt.title('Q-Q Plot', fontweight = 'bold', fontsize = 18, fontfamily = 'sans-serif', color = my_colors[1],pad=2
ax_3 = fig.add_subplot(1, 2, 1)
sns.boxenplot(y = 'bmi', data = df, color = "Gold", linewidth = 1.5);
plt.title('Boxen Plot', fontweight = 'bold', fontsize = 18, fontfamily = 'sans-serif', color = my_colors[1],pad
plt.tight_layout();
plt.show();
```



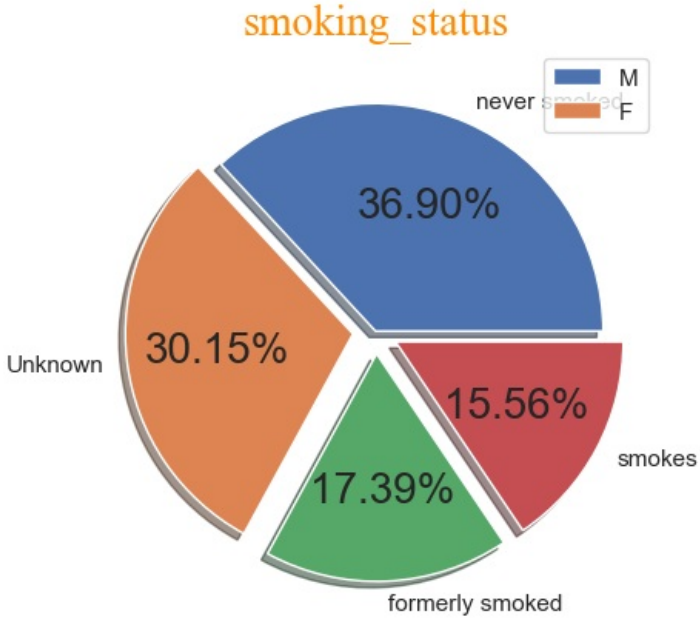




In [56]:
```
# Plot a subset of variables
g = sns.PairGrid(df, hue='hypertension' ,x_vars=["avg_glucose_level" , "bmi"],y_vars=["avg_glucose_level" , "bm
                 height=6, aspect=1)
g = g.map_offdiag(plt.scatter , edgecolor="w", s=130)
g = g.map_diag(plt.hist , edgecolor ='w', linewidth=2)
g = g.add_legend()
plt.show()
```

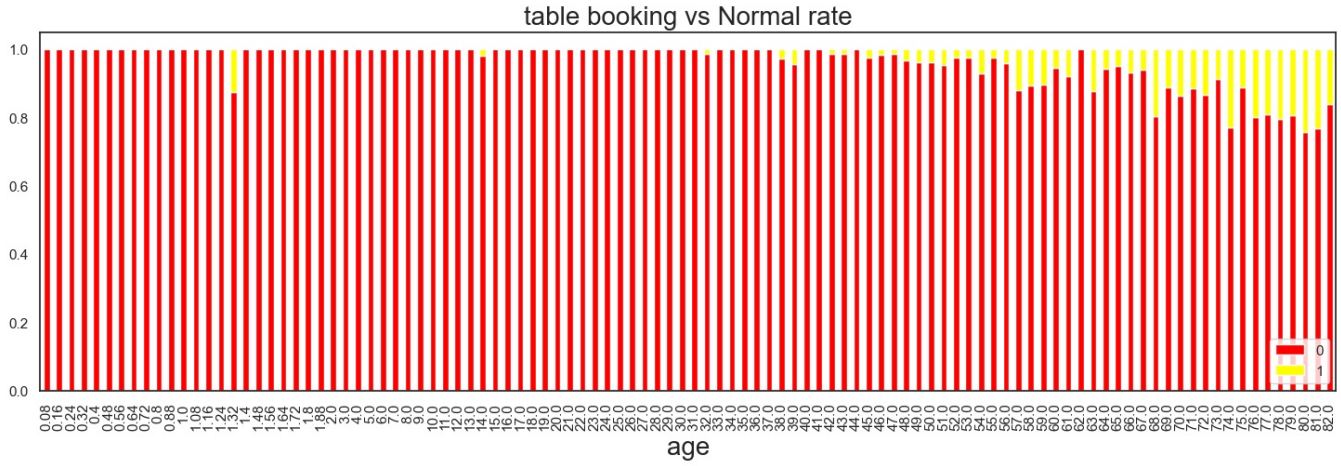


In [57]:
```
import matplotlib
matplotlib.rcParams.update({'font.size': 20})
ax=df['smoking_status'].value_counts().plot.pie(explode=[0.0, 0.1,0.1,0.1],autopct='%1.2f%%',shadow=True);
ax.set_title(label = "smoking_status", fontsize = 40,color='DarkOrange',font='Times New Roman');
plt.legend(labels=['M','F'])
plt.axis('off');
```

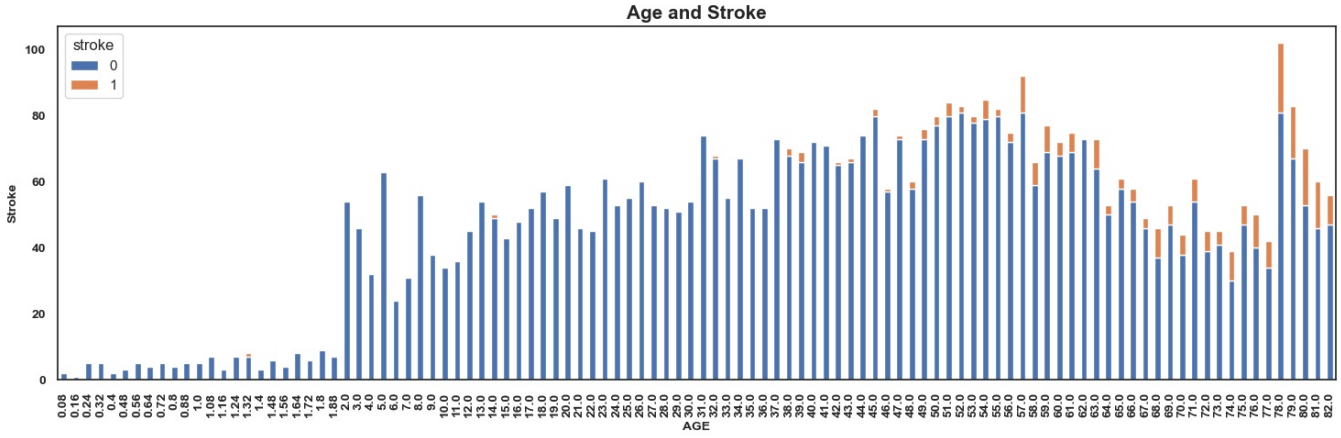


In [58]:
```
plt.rcParams['figure.figsize'] = (18, 5)
Y = pd.crosstab(df['age'], df['stroke'])
Y.div(Y.sum(1).astype(float), axis = 0).plot(kind = 'bar', stacked = True,color=['red','yellow'])
plt.title('table booking vs Normal rate', fontweight = 30, fontsize = 20)
```

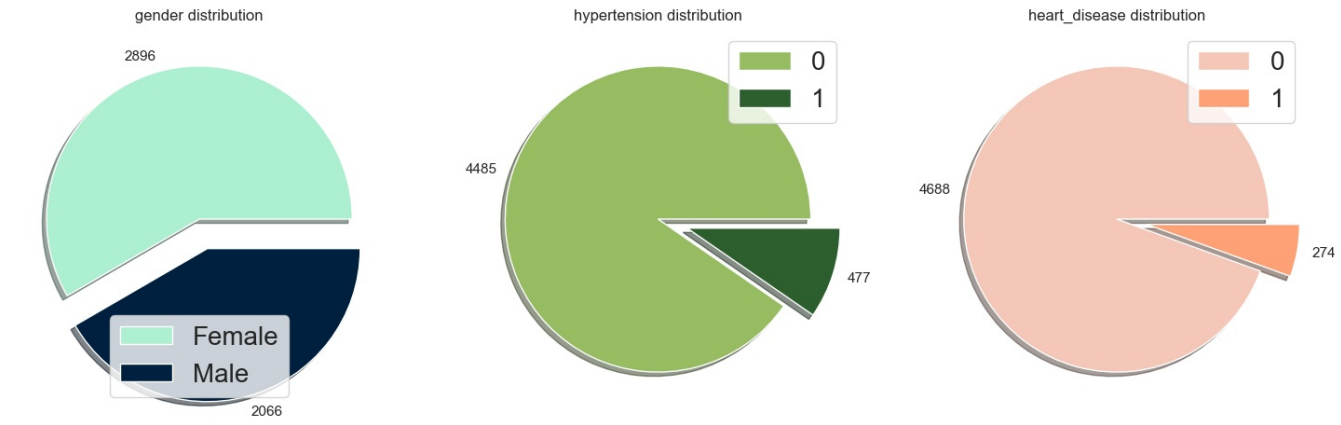
```
plt.legend(loc="lower right")
plt.show()
```



In [59]:
```
plt.rcParams['figure.figsize'] = (18, 5)
type_plt=pd.crosstab(df['age'],df['stroke'])
type_plt.plot(kind='bar',stacked=True);
plt.title('Age and Stroke',fontsize=15,fontweight='bold')
plt.ylabel('Stroke',fontsize=10,fontweight='bold')
plt.xlabel('AGE',fontsize=10,fontweight='bold')
plt.xticks(fontsize=10,fontweight='bold')
plt.yticks(fontsize=10,fontweight='bold');
```

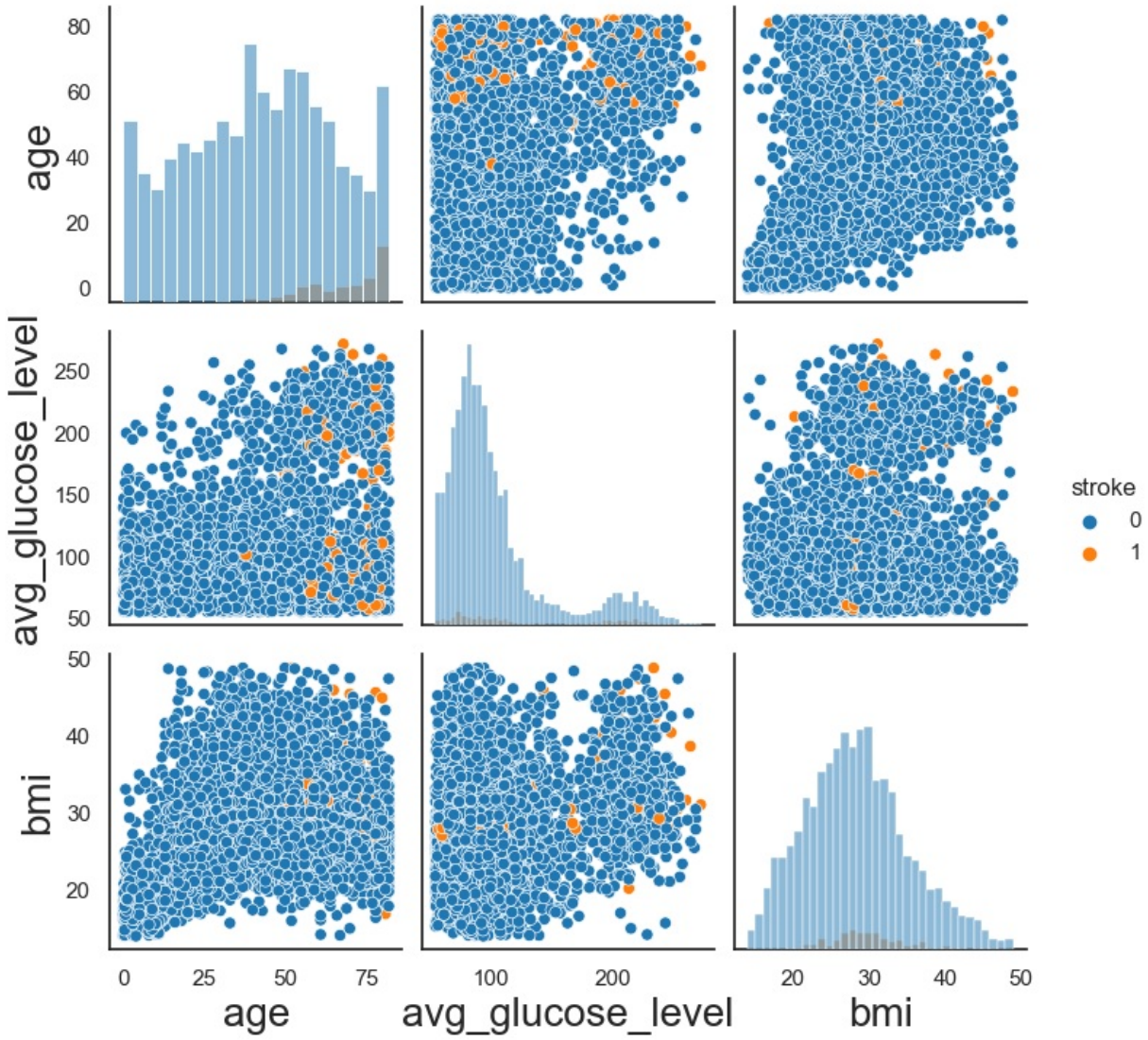


In [60]:
```
fig, ax = plt.subplots(ncols=3, figsize=(18,6))

colors = [['#ADEFD1FF', '#00203FFF'], ['#97BC62FF', '#2C5F2D'], ['#F5C7B8FF', '#FFA177FF']]
explode = [0, 0.2]
columns = ['gender', 'hypertension', 'heart_disease']
for i in range(3):
        data = df[columns[i]].value_counts()
        ax[i].pie(data, labels=data.values, explode=explode, colors=colors[i], shadow=True)
        ax[i].legend(labels=data.index, fontsize='medium')
        ax[i].set_title('{} distribution'.format(columns[i]))
```



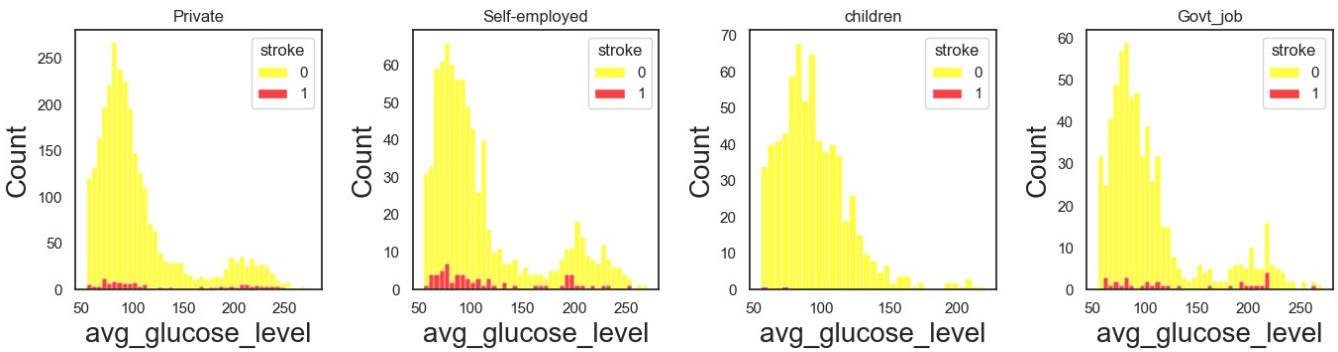
In [61]:
```
plt.figure(figsize=(18,9))
cols_out = ["age", "avg_glucose_level", "bmi","stroke"]
sns.pairplot(df[cols_out], hue="stroke", diag_kind="hist", palette="tab10") # tab10
plt.show();
```

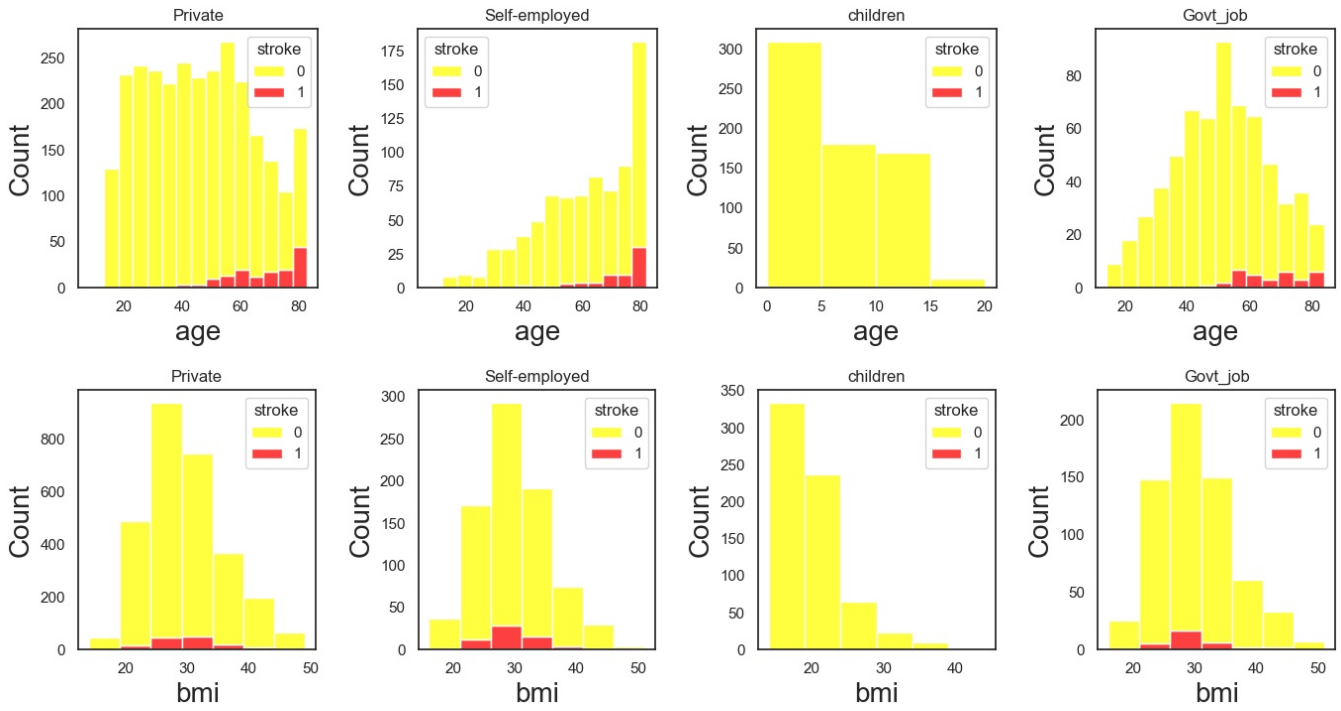
```
<Figure size 1800x900 with 0 Axes>
```



In [62]:

```
plot , ax = plt.subplots(1 , 4 , figsize=(14,4))
sns.histplot(data = df.loc[df["work_type"]=='Private'] , x = "avg_glucose_level" , hue = "stroke",binwidth=5,ax
sns.histplot(data = df.loc[df["work_type"]=='Self-employed'] , x = "avg_glucose_level" , hue = "stroke",binwidt
sns.histplot(data = df.loc[df["work_type"]=='children'] , x = "avg_glucose_level" , hue = "stroke",binwidth=5,a
sns.histplot(data = df.loc[df["work_type"]=='Govt_job'] , x = "avg_glucose_level" , hue = "stroke",binwidth=5,a
plt.tight_layout()
plt.show()

plot , ax = plt.subplots(1 , 4 , figsize=(14,4))
sns.histplot(data = df.loc[df["work_type"]=='Private'] , x = "age" , hue = "stroke",binwidth=5,ax = ax[0],palet
sns.histplot(data = df.loc[df["work_type"]=='Self-employed'] , x = "age" , hue = "stroke",binwidth=5,ax = ax[1]
sns.histplot(data = df.loc[df["work_type"]=='children'] , x = "age" , hue = "stroke",binwidth=5,ax = ax[2],pale
sns.histplot(data = df.loc[df["work_type"]=='Govt_job'] , x = "age" , hue = "stroke",binwidth=5,ax = ax[3],pale
plt.tight_layout()
plt.show()

plot , ax = plt.subplots(1 , 4 , figsize=(14,4))
sns.histplot(data = df.loc[df["work_type"]=='Private'] , x = "bmi" , hue = "stroke",binwidth=5,ax = ax[0],palet
sns.histplot(data = df.loc[df["work_type"]=='Self-employed'] , x = "bmi" , hue = "stroke",binwidth=5,ax = ax[1]
sns.histplot(data = df.loc[df["work_type"]=='children'] , x = "bmi" , hue = "stroke",binwidth=5,ax = ax[2],pale
sns.histplot(data = df.loc[df["work_type"]=='Govt_job'] , x = "bmi" , hue = "stroke",binwidth=5,ax = ax[3],pale
plt.tight_layout()
plt.show()
```





In [63]:
```
df = df.drop(['age_bins','avg_glucose_level_bins'],axis=1)
```

In [64]:
```
#for target feature
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['ever_married']=le.fit_transform(df['ever_married'])
df['work_type']=le.fit_transform(df['work_type'])
df['gender']=le.fit_transform(df['gender'])
df['work_type']=le.fit_transform(df['work_type'])
df['Residence_type']=le.fit_transform(df['Residence_type'])
df['smoking_status']=le.fit_transform(df['smoking_status'])
```

In [65]:
```
from sklearn.model_selection import train_test_split

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

X_train,X_test,y_train,y_test = train_test_split(X, y,test_size=0.3, random_state=21,shuffle=True,stratify=y)
```

In [66]:
```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
```

In [67]:
```
X_train, y_train = sm.fit_resample(X_train, y_train)
X_train.shape, y_train.shape
```

Out[67]:
```
((6600, 10), (6600,))
```

In [68]:
```
X_train.head(1)
```

Out[68]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
0	1	39.0	0	0	1	1	0	91.85	24.7	3

In [69]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [70]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import math
import re
plt.style.use('fivethirtyeight')
sns.set(context="notebook", palette="dark", style = 'whitegrid' , color_codes=True)
# Classification algorithms
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB, BernoulliNB
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, VotingClas
```

```
from sklearn.ensemble import HistGradientBoostingClassifier, ExtraTreesClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.neural_network import MLPClassifier
import lightgbm as lgb
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.gaussian_process.kernels import RBF
from sklearn.linear_model import Perceptron
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.svm import NuSVC
from sklearn.linear_model import PassiveAggressiveClassifier
from sklearn.experimental import enable_hist_gradient_boosting

names = ["Logistic Regression", "Random Forest","Naive Bayes GaussianNB","Naive Bayes_BernoulliNB","MLPClassifi
         "Nearest Neighbors","ExtraTreesClassifier","Linear SVC","rbf SVC","SGDClassifier","AdaBoost","Decision
         "Decision Tree gini","Decision Tree entropy","Gradient Boosting","LightGBM","LinearDiscriminantAnalysi
         "QuadraticDiscriminantAnalysis","Perceptron","PassiveAggressiveClassifier",
        "HistGradientBoosting"]

classifiers = [
LogisticRegression(),
RandomForestClassifier(criterion='entropy',n_estimators=100),
GaussianNB(),
BernoulliNB(),
MLPClassifier(),
KNeighborsClassifier(9),
ExtraTreesClassifier(n_estimators= 100),
SVC(kernel='linear',gamma='auto',probability=True),
SVC(kernel = "rbf", gamma=2, C=1),
SGDClassifier(max_iter=1000, tol=1e-4),
AdaBoostClassifier(),
DecisionTreeClassifier(),
DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=0),
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0),
GradientBoostingClassifier(n_estimators=100,max_features='sqrt'),
lgb.LGBMClassifier(),
LinearDiscriminantAnalysis(),
QuadraticDiscriminantAnalysis(),
Perceptron(),
PassiveAggressiveClassifier(random_state=0),
HistGradientBoostingClassifier()
]

accuracy_scores = []
# iterate over classifiers and predict accuracy
for name, clf in zip(names, classifiers):
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)
    score = round(score, 4)
    accuracy_scores.append(score)
    print(name ,' : ' , score)

classifiers_performance = pd.DataFrame({"Classifiers": names, "Accuracy Scores": accuracy_scores})

Best_score = classifiers_performance.sort_values(by = 'Accuracy Scores' , ascending = False)[['Classifiers', 'A
Best_score
```

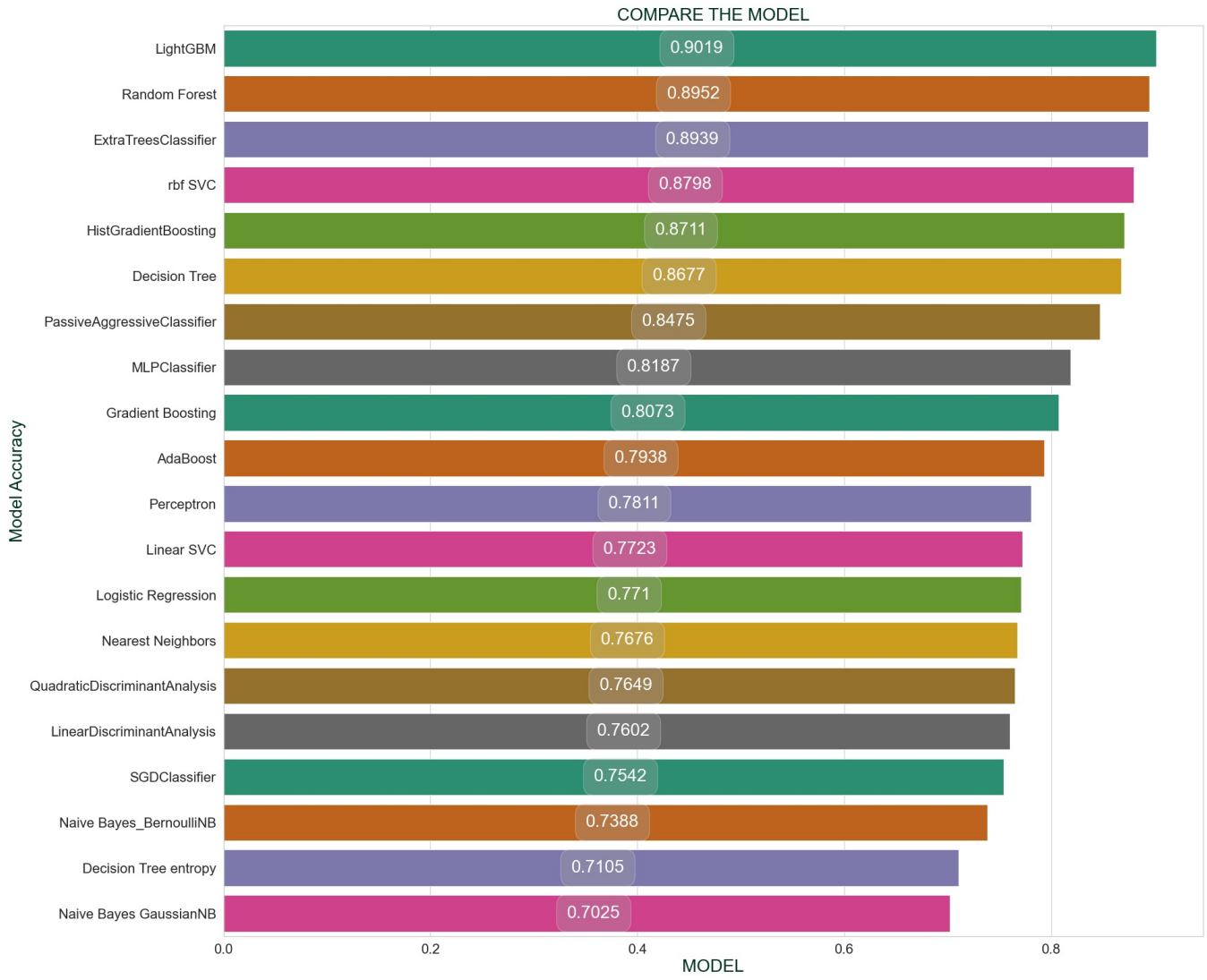
```
Logistic Regression  :  0.771
Random Forest  :  0.8952
Naive Bayes GaussianNB  :  0.7025
Naive Bayes_BernoulliNB  :  0.7388
MLPClassifier  :  0.8187
Nearest Neighbors  :  0.7676
ExtraTreesClassifier  :  0.8939
Linear SVC  :  0.7723
rbf SVC  :  0.8798
SGDClassifier  :  0.7542
AdaBoost  :  0.7938
Decision Tree  :  0.8677
Decision Tree gini  :  0.6897
Decision Tree entropy  :  0.7105
Gradient Boosting  :  0.8073
LightGBM  :  0.9019
LinearDiscriminantAnalysis  :  0.7602
QuadraticDiscriminantAnalysis  :  0.7649
Perceptron  :  0.7811
PassiveAggressiveClassifier  :  0.8475
HistGradientBoosting  :  0.8711
```

Out[70]:

	Classifiers	Accuracy Scores
15	LightGBM	0.9019
1	Random Forest	0.8952
6	ExtraTreesClassifier	0.8939
8	rbf SVC	0.8798
20	HistGradientBoosting	0.8711
11	Decision Tree	0.8677
19	PassiveAggressiveClassifier	0.8475
4	MLPClassifier	0.8187
14	Gradient Boosting	0.8073
10	AdaBoost	0.7938
18	Perceptron	0.7811
7	Linear SVC	0.7723
0	Logistic Regression	0.7710
5	Nearest Neighbors	0.7676
17	QuadraticDiscriminantAnalysis	0.7649
16	LinearDiscriminantAnalysis	0.7602
9	SGDClassifier	0.7542
3	Naive Bayes_BernoulliNB	0.7388
13	Decision Tree entropy	0.7105
2	Naive Bayes GaussianNB	0.7025
12	Decision Tree gini	0.6897

In [71]:

```
p = plt.figure(figsize=(18,18))
p = sns.set_palette('bright')
p = sns.set_context('paper', font_scale=1.8)

p = models=Best_score.sort_values(by='Accuracy Scores',ascending=False)[:20]

p = sns.barplot(y= 'Classifiers', x= 'Accuracy Scores', data= models,palette='Dark2')

for container in p.containers:
    p.bar_label(container,label_type = 'center',padding = 8,size = 20,color = "White",rotation = 0,
    bbox={"boxstyle": "round", "pad": 0.6, "facecolor": "#a9a9a9", "edgecolor": "white", "alpha": .3})

plt.title('COMPARE THE MODEL',fontsize=20,color='#013220')
plt.xlabel('MODEL',fontsize=20,color='#013220')
plt.ylabel('Model Accuracy',fontsize=20,color='#013220');
```



In [72]:

```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,f1_score
from sklearn.metrics import precision_score


Rf = RandomForestClassifier(n_jobs=5,criterion='entropy',n_estimators=100,verbose=False,random_state=51)
Rf.fit(X_train,y_train)

y_pred = Rf.predict(X_test)

print(f"\033[034m\033[1m")
print("Accuracy Score : ",round(accuracy_score(y_test,y_pred)*100,2),"%")
print("*******"*9)
print("Precision Score: ",round(precision_score(y_test,y_pred)*100,2),"%")
print("*******"*9)
print(f"\033[032m\033[1m")
print("F1 Score :",round(f1_score(y_test,y_pred)*100,2),"%")
print("*******"*9)
print(f"\033[035m\033[1m")
print("Confusin Matrix :\n",confusion_matrix(y_test,y_pred))
print(f"\033[031m\033[1m")
print("*******"*9)
print("Classification Report: \n",classification_report(y_test,y_pred))
```

```
Accuracy Score :  89.05 %
**********************************************************************
Precision Score:  12.61 %
**********************************************************************

F1 Score : 15.54 %
**********************************************************************

Confusin Matrix :
 [[1311  104]
 [  59   15]]

**********************************************************************
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.93      0.94      1415
           1       0.13      0.20      0.16        74

    accuracy                           0.89      1489
   macro avg       0.54      0.56      0.55      1489
weighted avg       0.92      0.89      0.90      1489
```

In [73]:
```
df = pd.read_csv("brain_stroke.csv")
print(f"\033[034m\033[1m")
print("Features Name :")
print(f"\033[031m\033[1m")
print(df.columns.tolist())
print(f"\033[035m\033[1m")
print("Data Shape :")
print(df.shape)

print(f"\033[034m\033[1m")
print("Data:")

df.head(10).style.set_properties(**{'background-color': 'Blue',
                           'color': 'white',
                           'border-color': 'darkblack'})
```

```
Features Name :

['gender', 'age', 'hypertension', 'heart_disease', 'ever_married', 'work_type', 'Residence_type', 'avg_glucose_
level', 'bmi', 'smoking_status', 'stroke']

Data Shape :
(4981, 11)

Data:
```

Out[73]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
0	Male	67.000000	0	1	Yes	Private	Urban	228.690000	36.600000	formerly smoked
1	Male	80.000000	0	1	Yes	Private	Rural	105.920000	32.500000	never smoked
2	Female	49.000000	0	0	Yes	Private	Urban	171.230000	34.400000	smokes
3	Female	79.000000	1	0	Yes	Self-employed	Rural	174.120000	24.000000	never smoked
4	Male	81.000000	0	0	Yes	Private	Urban	186.210000	29.000000	formerly smoked
5	Male	74.000000	1	1	Yes	Private	Rural	70.090000	27.400000	never smoked
6	Female	69.000000	0	0	No	Private	Urban	94.390000	22.800000	never smoked
7	Female	78.000000	0	0	Yes	Private	Urban	58.570000	24.200000	Unknown
8	Female	81.000000	1	0	Yes	Private	Rural	80.430000	29.700000	never smoked
9	Female	61.000000	0	1	Yes	Govt_job	Rural	120.460000	36.800000	smokes

In [74]:
```
#pip install pycaret
```

Setting Up PyCaret

In [75]:
```
import pycaret
from pycaret.classification import *
```

In [76]:
```
# --- Setting Up PyCaret ---
stp = setup(data = df, target = 'stroke', train_size = 0.8,
            categorical_features = ['gender', 'ever_married', 'work_type', 'Residence_type',
       'smoking_status'], fix_imbalance = True, session_id=123)
```

	Description	Value
0	Session id	123
1	Target	stroke
2	Target type	Binary
3	Original data shape	(4981, 11)
4	Transformed data shape	(8569, 17)
5	Transformed train set shape	(7572, 17)
6	Transformed test set shape	(997, 17)
7	Ordinal features	3
8	Numeric features	5
9	Categorical features	5
10	Rows with missing values	0.4%
11	Preprocess	True
12	Imputation type	simple
13	Numeric imputation	mean
14	Categorical imputation	mode
15	Maximum one-hot encoding	25
16	Encoding method	None
17	Fix imbalance	True
18	Fix imbalance method	SMOTE
19	Fold Generator	StratifiedKFold
20	Fold Number	10
21	CPU Jobs	-1
22	Use GPU	False
23	Log Experiment	False
24	Experiment Name	clf-default-name
25	USI	1180

In [77]:
```
best_model = compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
dummy	Dummy Classifier	0.9503	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.4240
gbc	Gradient Boosting Classifier	0.9448	0.7993	0.0100	0.0750	0.0174	0.0064	0.0108	1.6660
lightgbm	Light Gradient Boosting Machine	0.9445	0.8065	0.0350	0.1450	0.0563	0.0417	0.0516	1.4290
rf	Random Forest Classifier	0.9440	0.7801	0.0300	0.1475	0.0486	0.0342	0.0458	1.2640
ada	Ada Boost Classifier	0.9435	0.8176	0.0613	0.1917	0.0900	0.0730	0.0860	0.8550
catboost	CatBoost Classifier	0.9428	0.7938	0.0300	0.1293	0.0484	0.0320	0.0409	7.3630
et	Extra Trees Classifier	0.9393	0.7586	0.0555	0.2037	0.0829	0.0603	0.0766	1.1240
xgboost	Extreme Gradient Boosting	0.9345	0.7920	0.0250	0.0586	0.0348	0.0090	0.0088	1.4860
dt	Decision Tree Classifier	0.9041	0.5354	0.1261	0.0970	0.1080	0.0585	0.0598	0.3500
knn	K Neighbors Classifier	0.7972	0.7198	0.5255	0.1291	0.2068	0.1379	0.1828	0.4400
lr	Logistic Regression	0.7420	0.8325	0.7929	0.1378	0.2345	0.1637	0.2561	2.7440
ridge	Ridge Classifier	0.7334	0.0000	0.8029	0.1352	0.2312	0.1597	0.2541	0.3670
lda	Linear Discriminant Analysis	0.7334	0.8342	0.8029	0.1352	0.2312	0.1597	0.2541	0.4130
svm	SVM - Linear Kernel	0.7183	0.0000	0.7366	0.1269	0.2083	0.1369	0.2204	0.4400
qda	Quadratic Discriminant Analysis	0.6739	0.7808	0.7237	0.1045	0.1821	0.1044	0.1816	0.3870
nb	Naive Bayes	0.6150	0.7844	0.8232	0.0981	0.1753	0.0949	0.1885	0.3550

```
Processing:   0%|          | 0/69 [00:00<?, ?it/s]
```

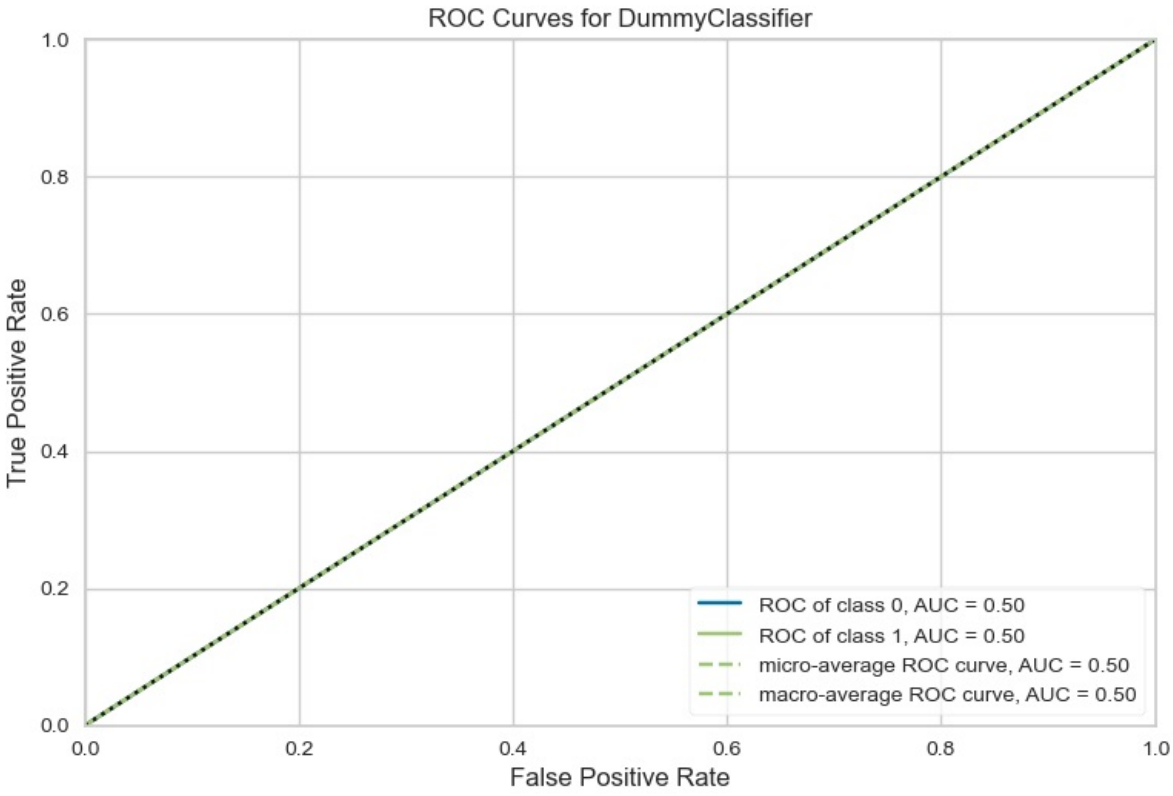
In [78]:
```
print(best_model)
```

```
DummyClassifier(constant=None, random_state=123, strategy='prior')
```

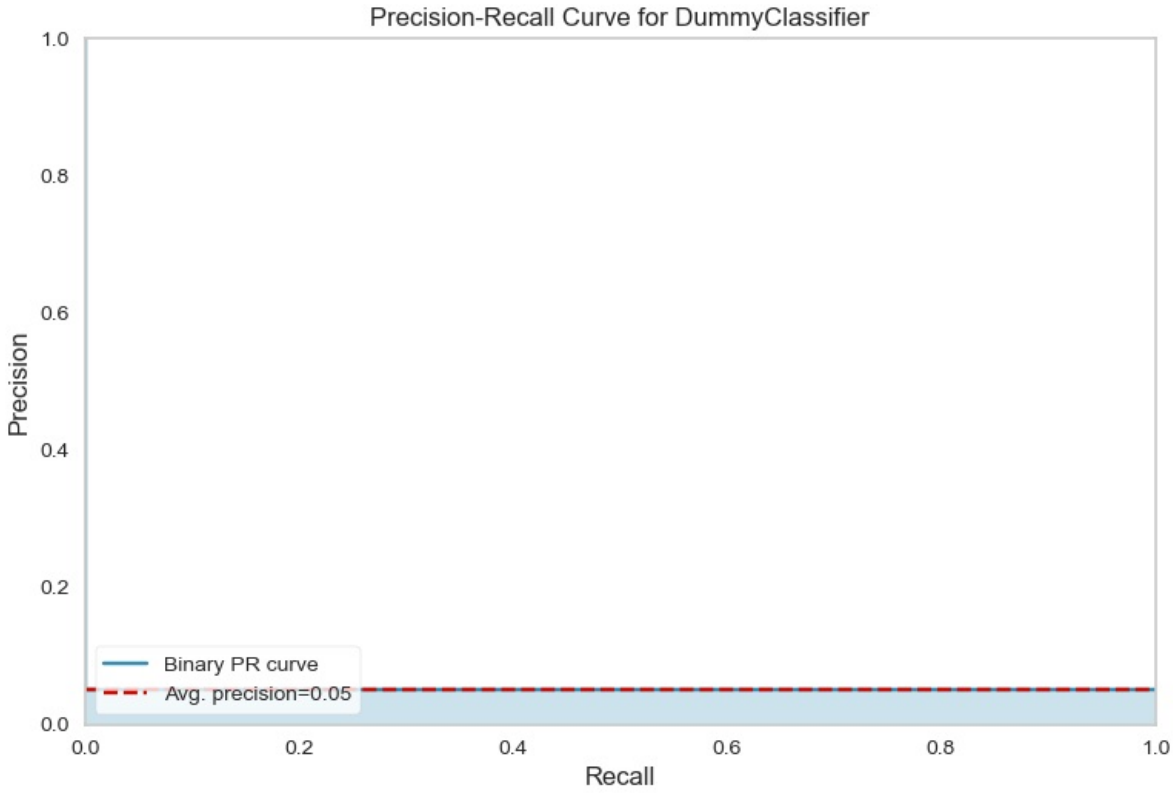
In [79]:
```
evaluate_model(best_model)
```

```
interactive(children=(ToggleButtons(description='Plot Type:', icons=('',), options=(('Pipeline Plot', 'pipelin…
```

In [80]:
```
# --- Plotting AUC plot ---
plot_model(best_model, plot = 'auc')
```



In [81]:
```
# --- Plotting Precision-Recall Curve ---
plot_model(best_model, plot = 'pr')
```



In [82]:
```
# --- Create Logistic Regression Model ---
logreg  = create_model('lr')
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7544	0.8489	0.8000	0.1455	0.2462	0.1763	0.2695
1	0.7544	0.8015	0.7500	0.1389	0.2344	0.1636	0.2478
2	0.7419	0.8584	0.8500	0.1453	0.2482	0.1778	0.2809
3	0.7343	0.8016	0.7500	0.1293	0.2206	0.1477	0.2323
4	0.7462	0.8109	0.7895	0.1339	0.2290	0.1605	0.2530
5	0.7261	0.8514	0.7895	0.1250	0.2158	0.1454	0.2381
6	0.7236	0.8753	0.8500	0.1371	0.2361	0.1637	0.2674
7	0.7940	0.8340	0.7000	0.1556	0.2545	0.1878	0.2606
8	0.7286	0.8192	0.9000	0.1452	0.2500	0.1789	0.2923
9	0.7161	0.8238	0.7500	0.1220	0.2098	0.1350	0.2195
Mean	0.7420	0.8325	0.7929	0.1378	0.2345	0.1637	0.2561
Std	0.0213	0.0240	0.0568	0.0099	0.0147	0.0161	0.0214

```
Processing:   0%|          | 0/4 [00:00<?, ?it/s]
```

In [83]:
```
# --- Tuning Logistic Regression ---
tune_lr = tune_model(logreg)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7569	0.8485	0.8000	0.1468	0.2481	0.1785	0.2716
1	0.7569	0.8022	0.7500	0.1402	0.2362	0.1658	0.2498
2	0.7419	0.8583	0.8500	0.1453	0.2482	0.1778	0.2809
3	0.7393	0.8013	0.7500	0.1316	0.2239	0.1515	0.2361
4	0.7462	0.8100	0.7895	0.1339	0.2290	0.1605	0.2530
5	0.7286	0.8510	0.7895	0.1261	0.2174	0.1472	0.2399
6	0.7236	0.8741	0.8500	0.1371	0.2361	0.1637	0.2674
7	0.7940	0.8328	0.7000	0.1556	0.2545	0.1878	0.2606
8	0.7286	0.8187	0.9000	0.1452	0.2500	0.1789	0.2923
9	0.7161	0.8241	0.7500	0.1220	0.2098	0.1350	0.2195
Mean	0.7432	0.8321	0.7929	0.1384	0.2353	0.1647	0.2571
Std	0.0213	0.0238	0.0568	0.0098	0.0143	0.0157	0.0209

```
Processing:   0%|          | 0/7 [00:00<?, ?it/s]
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

In [84]:
```
# --- Prediction on Test Sample w/ Tuned LR ---
predict_model(tune_lr);
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Logistic Regression	0.7302	0.8451	0.7800	0.1313	0.2248	0.1520	0.2422

In [85]:
```
# --- Finalize Tuned Logistic Regression ---
model_final = finalize_model(tune_lr)
print(model_final)
```

```
Pipeline(memory=FastMemory(location=C:\Users\pytho\AppData\Local\Temp\joblib),
         steps=[('numerical_imputer',
                 TransformerWrapper(exclude=None,
                                    include=['age', 'hypertension',
                                             'heart_disease',
                                             'avg_glucose_level', 'bmi'],
                                    transformer=SimpleImputer(add_indicator=False,
                                                              copy=True,
                                                              fill_value=None,
                                                              keep_empty_features=False,
                                                              missing_values=nan,
                                                              strategy='mean',
                                                              verbos...
                                    transformer=FixImbalancer(estimator=SMOTE(k_neighbors=5,
                                                                              n_jobs=None,
                                                                              random_state=None,
                                                                              sampling_strategy='auto')))),
                ('actual_estimator',
                 LogisticRegression(C=8.866, class_weight='balanced',
                                    dual=False, fit_intercept=True,
                                    intercept_scaling=1, l1_ratio=None,
                                    max_iter=1000, multi_class='auto',
                                    n_jobs=None, penalty='l2', random_state=123,
                                    solver='lbfgs', tol=0.0001, verbose=0,
                                    warm_start=False))],
         verbose=False)
```