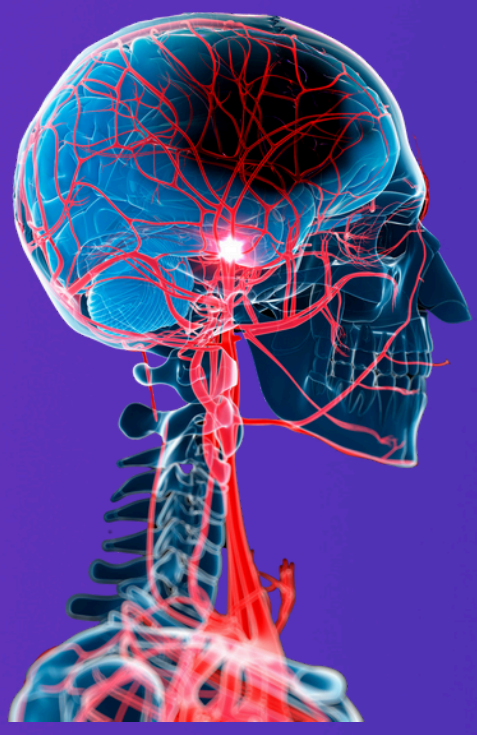
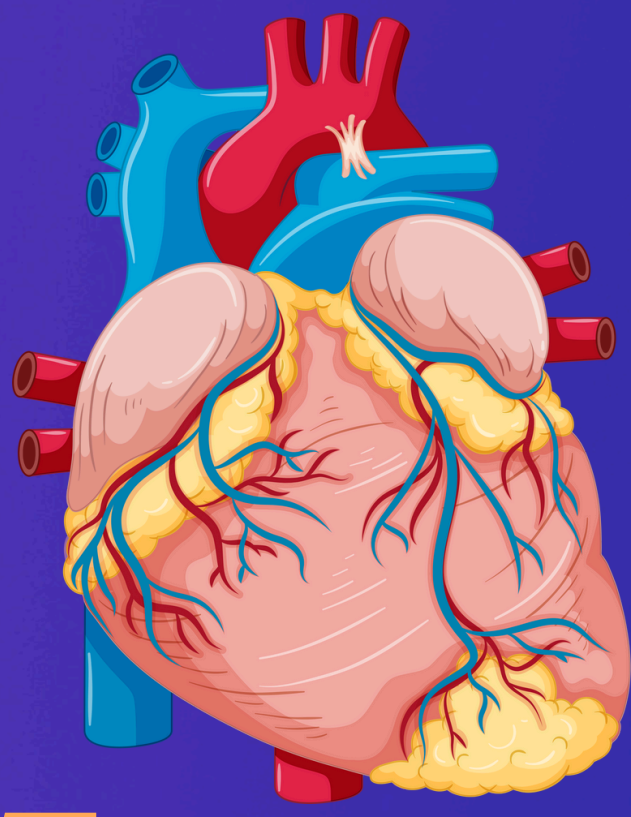
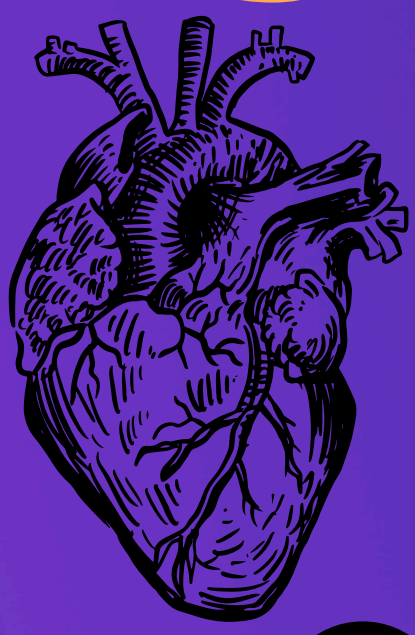




Stroke Prediction.



ML (classification)!

CREATED BY

UDAYABHANU NAYAK

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
df = pd.read_csv(r'C:\Users\Udayabhanu\Downloads\Stroke Prediction\Stroke Data
```

In [3]:
```
df.head()
```

Out[3]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type
0	9046	Male	67.0	0	1	Yes	Private	Urban
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural
2	31112	Male	80.0	0	1	Yes	Private	Rural
3	60182	Female	49.0	0	0	Yes	Private	Urban
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural

In [4]:
```
df.tail()
```

Out[4]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_ty
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

In [5]:
```
df = df.drop('id',axis=1)
df
```

Out[5]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg
0	Male	67.0	0	1	Yes	Private	Urban	
1	Female	61.0	0	0	Yes	Self-employed	Rural	
2	Male	80.0	0	1	Yes	Private	Rural	
3	Female	49.0	0	0	Yes	Private	Urban	
4	Female	79.0	1	0	Yes	Self-employed	Rural	
...	...	...	...	...	...	...	...	...
5105	Female	80.0	1	0	Yes	Private	Urban	
5106	Female	81.0	0	0	Yes	Self-employed	Urban	
5107	Female	35.0	0	0	Yes	Self-employed	Rural	
5108	Male	51.0	0	0	Yes	Private	Rural	
5109	Female	44.0	0	0	Yes	Govt_job	Urban	

5110 rows × 11 columns

In [6]:
```
df.shape
```

Out[6]:
```
(5110, 11)
```

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   gender             5110 non-null   object 
 1   age                5110 non-null   float64
 2   hypertension       5110 non-null   int64  
 3   heart_disease      5110 non-null   int64  
 4   ever_married       5110 non-null   object 
 5   work_type          5110 non-null   object 
 6   Residence_type     5110 non-null   object 
 7   avg_glucose_level  5110 non-null   float64
 8   bmi                4909 non-null   float64
 9   smoking_status     5110 non-null   object 
 10  stroke             5110 non-null   int64  
dtypes: float64(3), int64(3), object(5)
memory usage: 439.3+ KB
```

In [8]:
```
df.describe()
```

Out[8]:

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

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                201
smoking_status       0
stroke               0
dtype: int64
```

In [10]:
```
df.isnull().sum().sort_values(ascending=False)
```

Out[10]:
```
bmi                201
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
smoking_status       0
stroke               0
dtype: int64
```

In [11]:
```
df.nunique()
```

Out[11]:
```
gender                 3
age                  104
hypertension           2
heart_disease          2
ever_married           2
work_type              5
Residence_type         2
avg_glucose_level    3979
bmi                   418
smoking_status          4
stroke                  2
dtype: int64
```

In [12]:
```
df.nunique().sort_values(ascending=False)
```

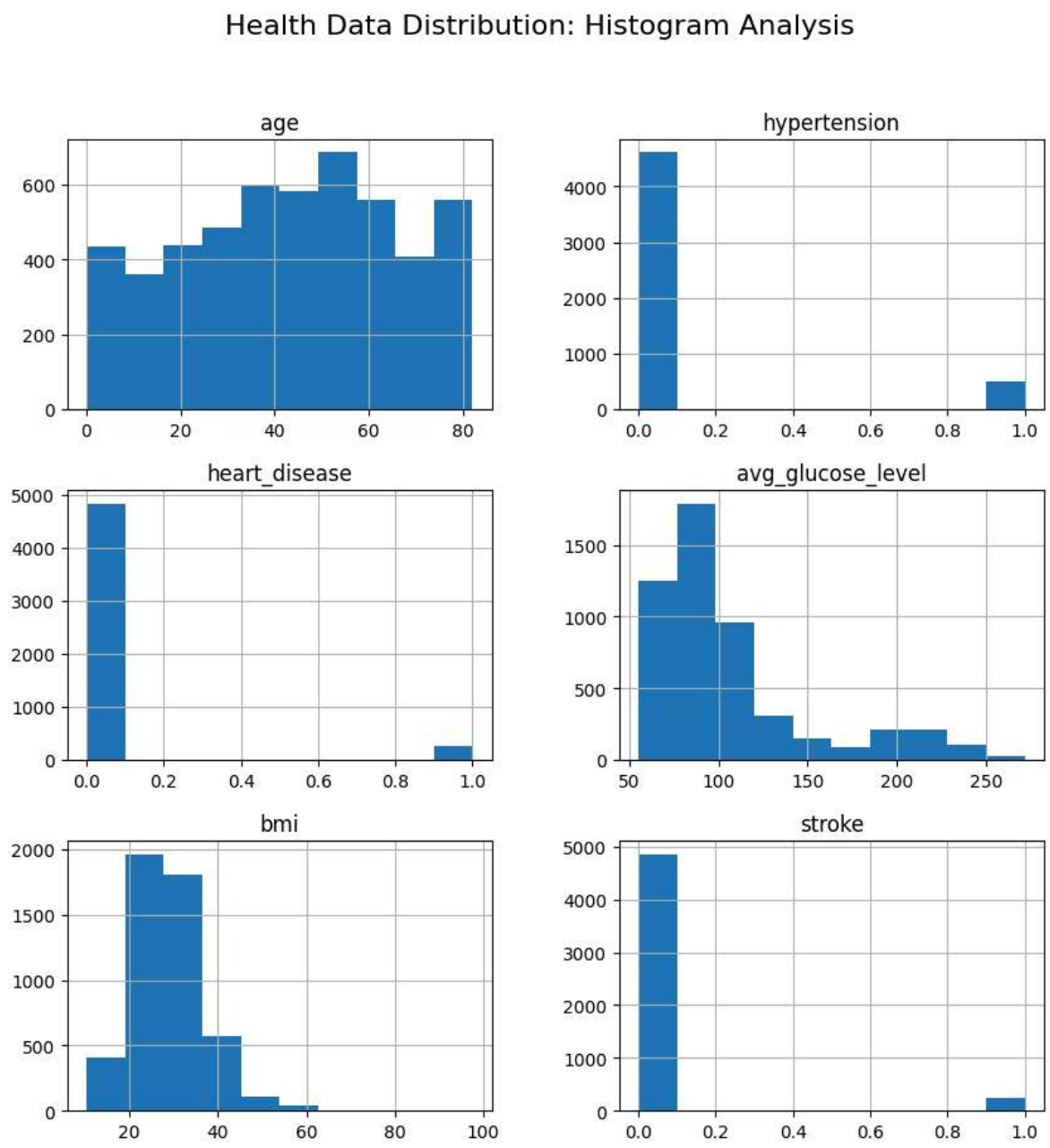
Out[12]:
```
avg_glucose_level    3979
bmi                   418
age                   104
work_type               5
smoking_status          4
gender                  3
hypertension            2
heart_disease           2
ever_married            2
Residence_type          2
stroke                  2
dtype: int64
```

In [13]:
```
df.duplicated().sum()
```

Out[13]:
```
0
```

In [14]:
```
import matplotlib.pyplot as plt
```

In [15]:
```
df.hist(figsize=(10,10))
plt.suptitle('Health Data Distribution: Histogram Analysis', fontsize=16)
plt.show()
```



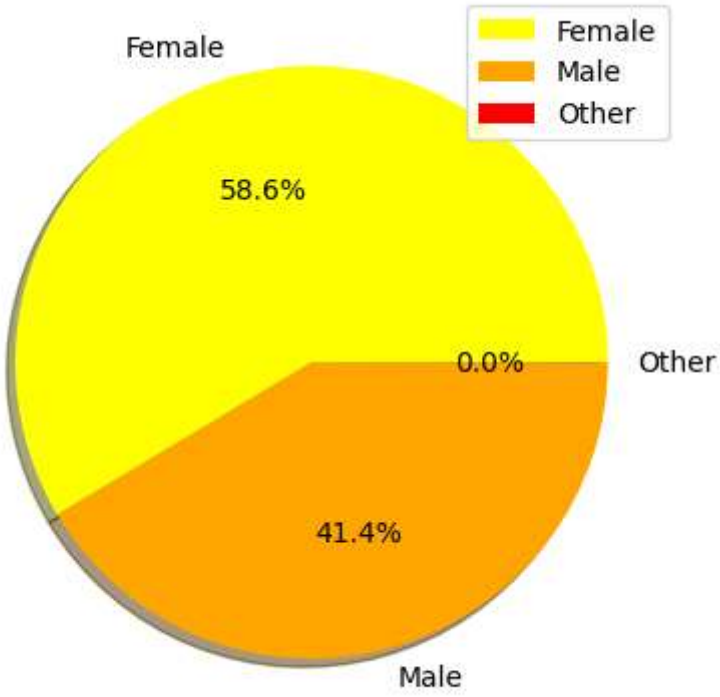
Gender Analysis

In [17]:
```
gender_count=df['gender'].value_counts()
gender_percent=df['gender'].value_counts(normalize=True)*100
table1=pd.DataFrame({'Count':gender_count,'Percentage %':gender_percent})
table1
```

Out[17]:

gender	Count	Percentage %
Female	2994	58.590998
Male	2115	41.389432
Other	1	0.019569

In [18]:
```
gender_counts = df['gender'].value_counts()
plt.pie(gender_counts, labels=gender_counts.index, colors=['yellow', 'orange'
plt.legend()
plt.show()
```

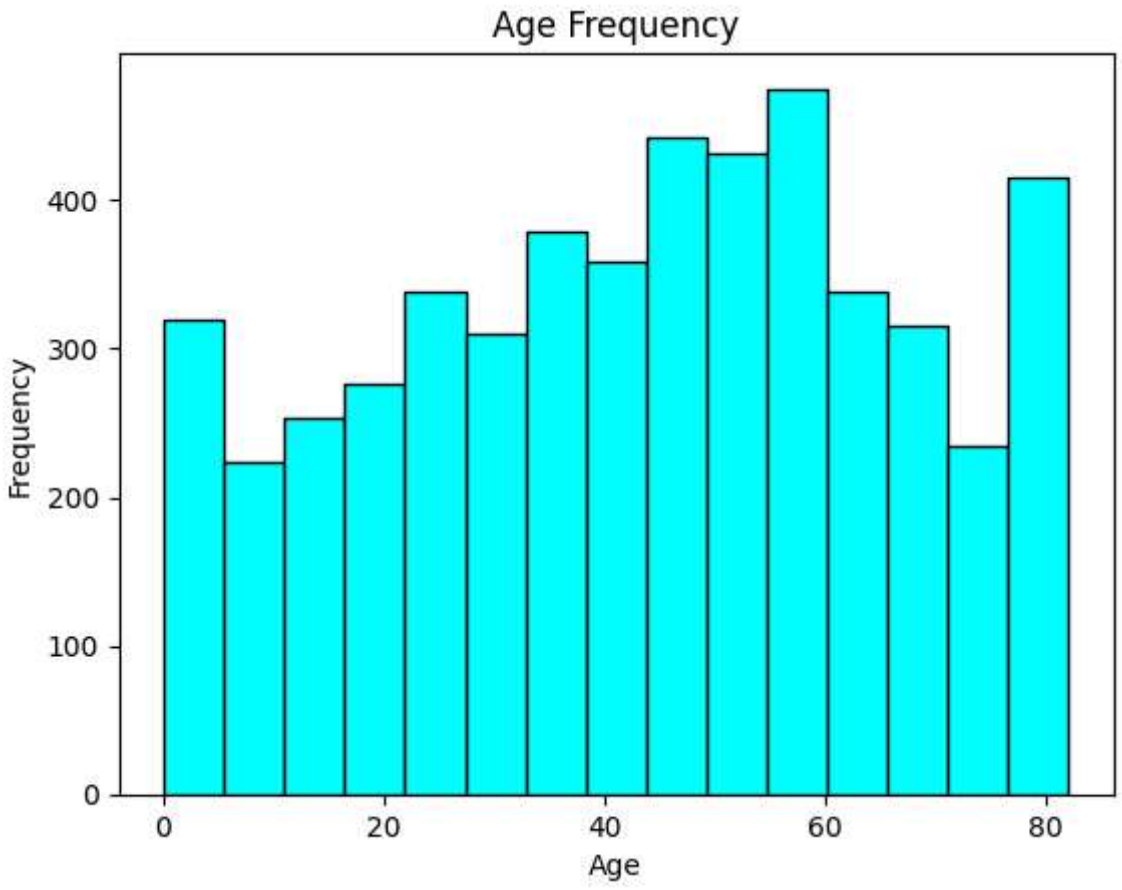


Age Analysis

In [20]:
```
df['age'].describe()
```

Out[20]:
```
count    5110.000000
mean       43.226614
std        22.612647
min         0.080000
25%        25.000000
50%        45.000000
75%        61.000000
max        82.000000
Name: age, dtype: float64
```

In [21]:
```
plt.hist(df['age'],bins=15,color='cyan',edgecolor='black',linewidth=1)
plt.title('Age Frequency')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



In [22]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   gender             5110 non-null   object 
 1   age                5110 non-null   float64
 2   hypertension       5110 non-null   int64  
 3   heart_disease      5110 non-null   int64  
 4   ever_married       5110 non-null   object 
 5   work_type          5110 non-null   object 
 6   Residence_type     5110 non-null   object 
 7   avg_glucose_level  5110 non-null   float64
 8   bmi                4909 non-null   float64
 9   smoking_status     5110 non-null   object 
 10  stroke             5110 non-null   int64  
dtypes: float64(3), int64(3), object(5)
memory usage: 439.3+ KB
```

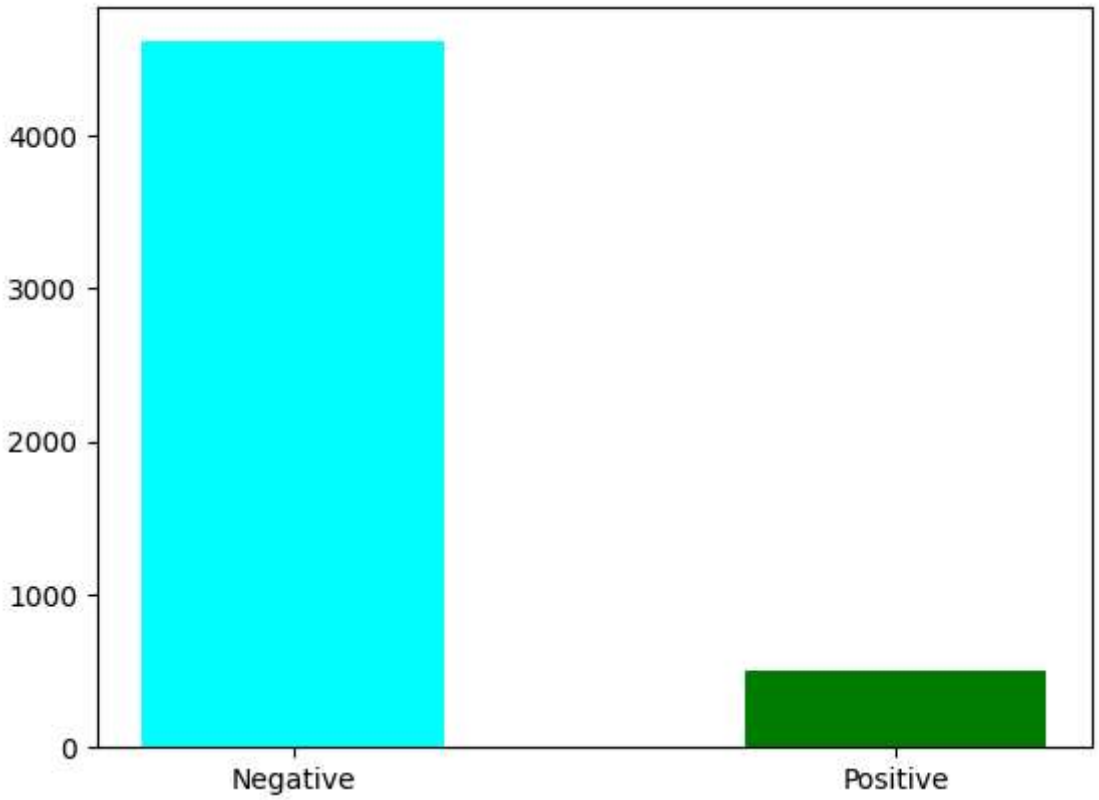
Hyper Tension Analysis

In [24]:
```
hyperTension_count=df['hypertension'].value_counts()
hyperTension_percent=df['hypertension'].value_counts(normalize=True)*100
table2=pd.DataFrame({'Count':hyperTension_count,'Percentage %':hyperTension_p
table2
```

Out[24]:

hypertension	Count	Percentage %
0	4612	90.254403
1	498	9.745597

In [25]:
```
plt.bar(['Negative','Positive'],df['hypertension'].value_counts(),align='cent
plt.show()
```



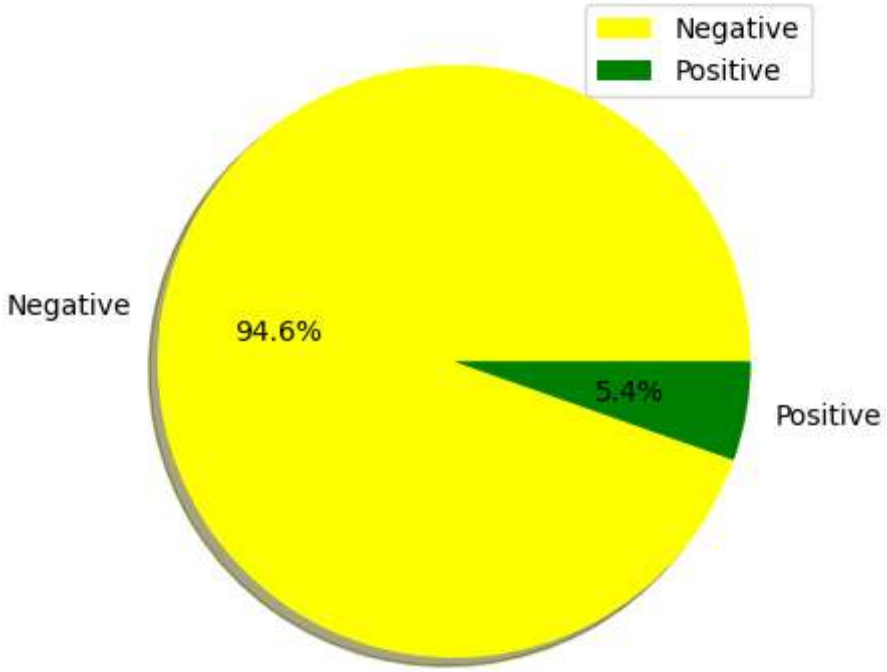
Heart Disease Analysis

In [27]:
```
heartDisease_count=df['heart_disease'].value_counts()
heartDisease_percent=df['heart_disease'].value_counts(normalize=True)*100
table3=pd.DataFrame({'Count':heartDisease_count,'Percantage %':heartDisease_p
table3
```

Out[27]:

heart_disease	Count	Percantage %
0	4834	94.598826
1	276	5.401174

In [28]:
```
plt.pie(df['heart_disease'].value_counts(),labels=['Negative','Positive'],col
plt.legend()
plt.show()
```



Marriage Analysis

In [30]:
```
df[['ever_married']]
```

Out[30]:

	ever_married
0	Yes
1	Yes
2	Yes
3	Yes
4	Yes
...	...
5105	Yes
5106	Yes
5107	Yes
5108	Yes
5109	Yes

5110 rows × 1 columns

In [31]:
```
married_count=df['ever_married'].value_counts()
married_percent=df['ever_married'].value_counts(normalize=True)*100
table4=pd.DataFrame({'Count':married_count,'Percentage %':married_percent})
table4
```

Out[31]:

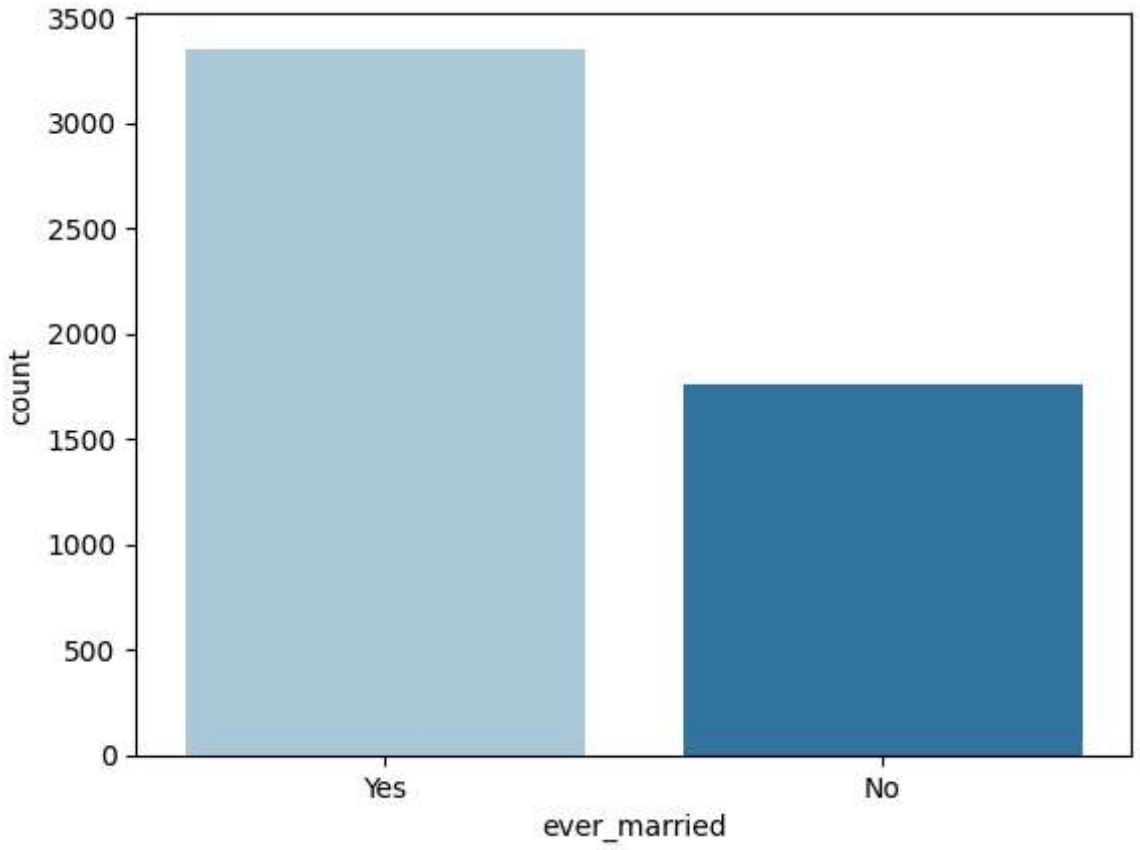
ever_married	Count	Percentage %
Yes	3353	65.616438
No	1757	34.383562

In [32]:
```
import seaborn as sns
```

In [33]:
```
sns.countplot(x='ever_married',palette=sns.color_palette("Paired"),data=df)
```

```
C:\Users\Udayabhanu\AppData\Local\Temp\ipykernel_12612\3970370607.py:1: Futu
reWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the
same effect.

  sns.countplot(x='ever_married',palette=sns.color_palette("Paired"),data=d
f)
C:\Users\Udayabhanu\AppData\Local\Temp\ipykernel_12612\3970370607.py:1: User
Warning: The palette list has more values (12) than needed (2), which may no
t be intended.
  sns.countplot(x='ever_married',palette=sns.color_palette("Paired"),data=d
f)
```

Out[33]: <Axes: xlabel='ever_married', ylabel='count'>



Work Type Analysis

In [35]:
```
job_count=df['work_type'].value_counts()
job_percent=df['work_type'].value_counts(normalize=True)*100
table5=pd.DataFrame({'Count':job_count,'Percentage %':job_percent})
table5
```

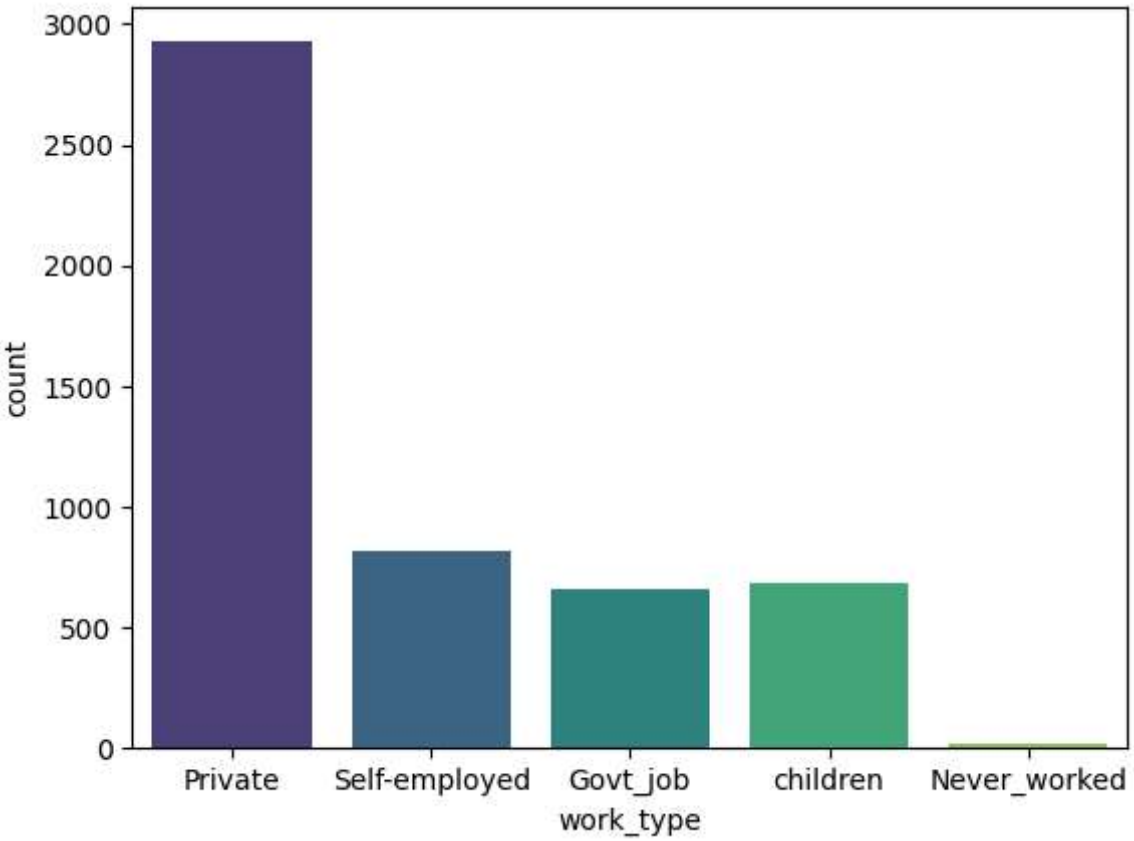
Out[35]:

work_type	Count	Percentage %
Private	2925	57.240705
Self-employed	819	16.027397
children	687	13.444227
Govt_job	657	12.857143
Never_worked	22	0.430528

In [36]:
```
sns.countplot(x='work_type',palette='viridis',data=df)
```

```
C:\Users\Udayabhanu\AppData\Local\Temp\ipykernel_12612\3879642141.py:1: Futu
reWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the
same effect.

  sns.countplot(x='work_type',palette='viridis',data=df)
```

Out[36]: <Axes: xlabel='work_type', ylabel='count'>



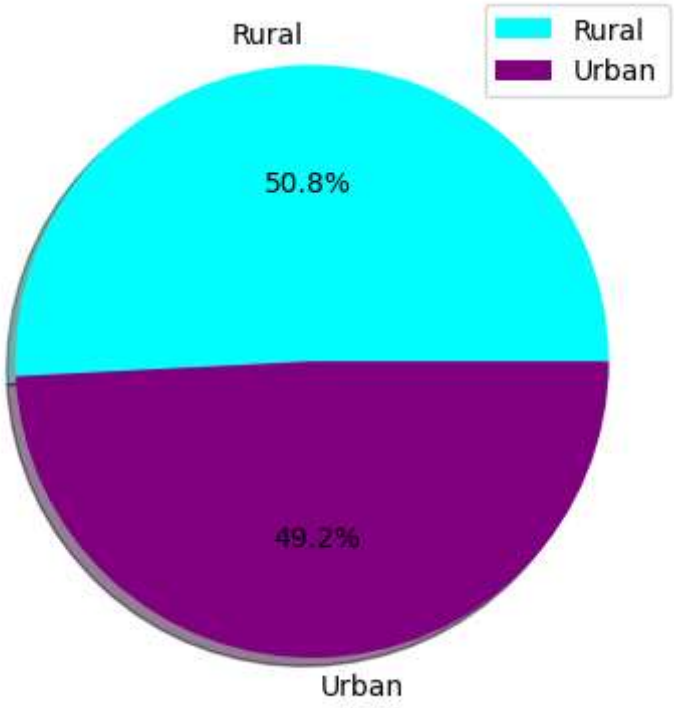
Residence Type Analysis

In [38]:
```
Residence_type_count=df['Residence_type'].value_counts()
Residence_type_percent=df['Residence_type'].value_counts(normalize=True)*100
table6=pd.DataFrame({'Count':Residence_type_count,'Percentage %':Residence_ty
table6
```

Out[38]:

Residence_type	Count	Percentage %
Urban	2596	50.802348
Rural	2514	49.197652

In [39]:
```
plt.pie(df['Residence_type'].value_counts(),labels=['Rural','Urban'],colors=[
plt.legend()
plt.show()
```

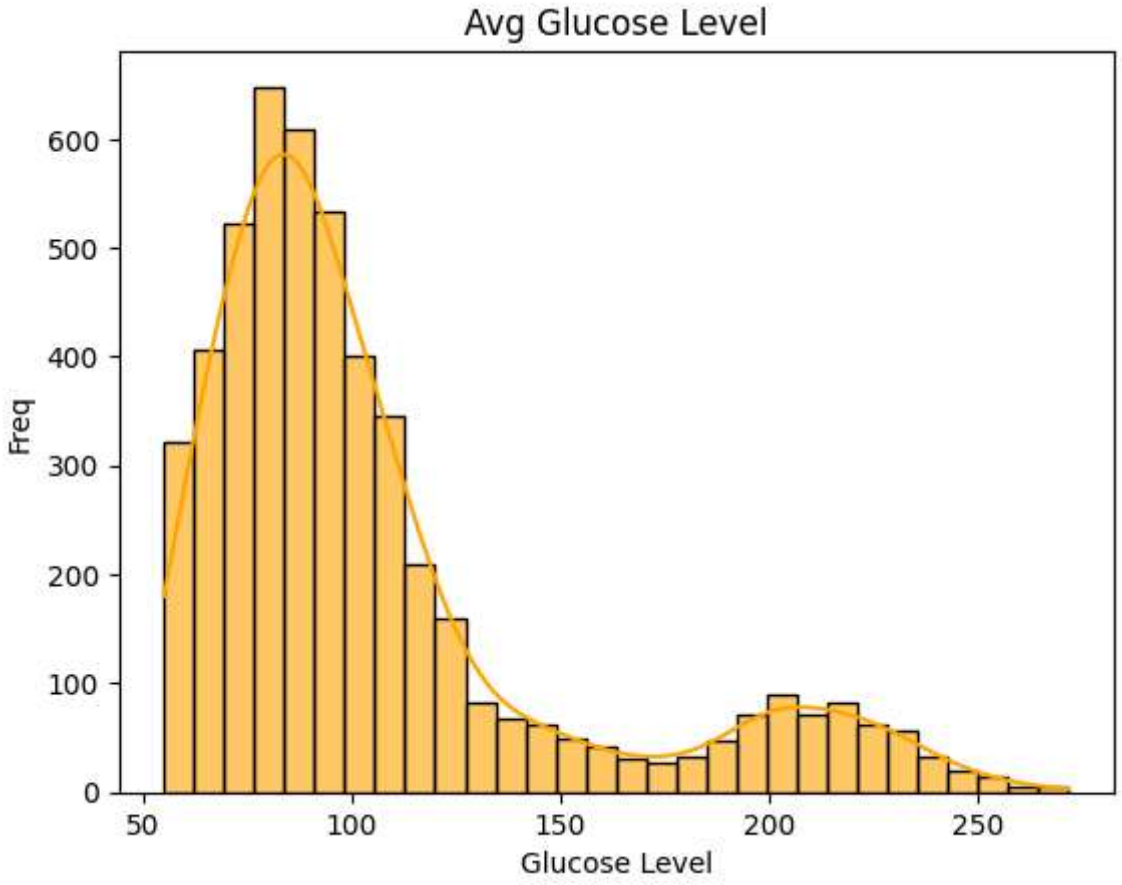


Average Glucose Level Analysis

In [41]:
```
df['avg_glucose_level'].describe()
```

Out[41]:
```
count    5110.000000
mean      106.147677
std        45.283560
min        55.120000
25%        77.245000
50%        91.885000
75%       114.090000
max       271.740000
Name: avg_glucose_level, dtype: float64
```

In [42]:
```
sns.histplot(df['avg_glucose_level'],bins=30,kde=True,color='orange',edgecolo
plt.title('Avg Glucose Level')
plt.xlabel('Glucose Level')
plt.ylabel('Freq')
plt.show()
```



Body Mass Index Analysis

In [44]:
```
df['bmi'].describe()
```

Out[44]:
```
count    4909.000000
mean       28.893237
std         7.854067
min        10.300000
25%        23.500000
50%        28.100000
75%        33.100000
max        97.600000
Name: bmi, dtype: float64
```

In [45]:
```
df['bmi'].isnull().sum()
```

Out[45]:
```
201
```

In [46]:
```
df.dropna(inplace=True)
```

In [47]:
```
df[df['bmi'].isna()]
```

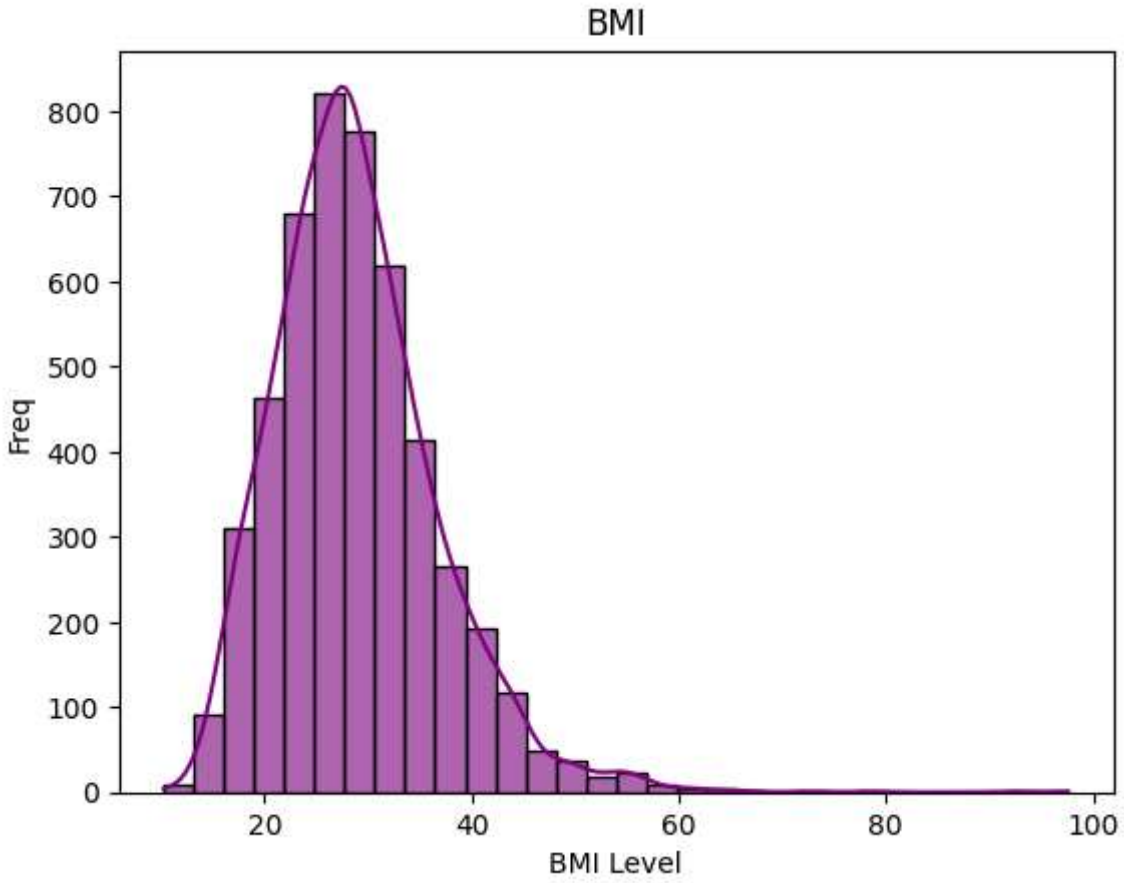
Out[47]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_gluc

In [48]:
```
df['bmi'].describe()
```

Out[48]:
```
count    4909.000000
mean       28.893237
std         7.854067
min        10.300000
25%        23.500000
50%        28.100000
75%        33.100000
max        97.600000
Name: bmi, dtype: float64
```

In [49]:
```
sns.histplot(df['bmi'],bins=30,kde=True,color='purple',edgecolor='black',linew
plt.title('BMI')
plt.xlabel('BMI Level')
plt.ylabel('Freq')
plt.show()
```



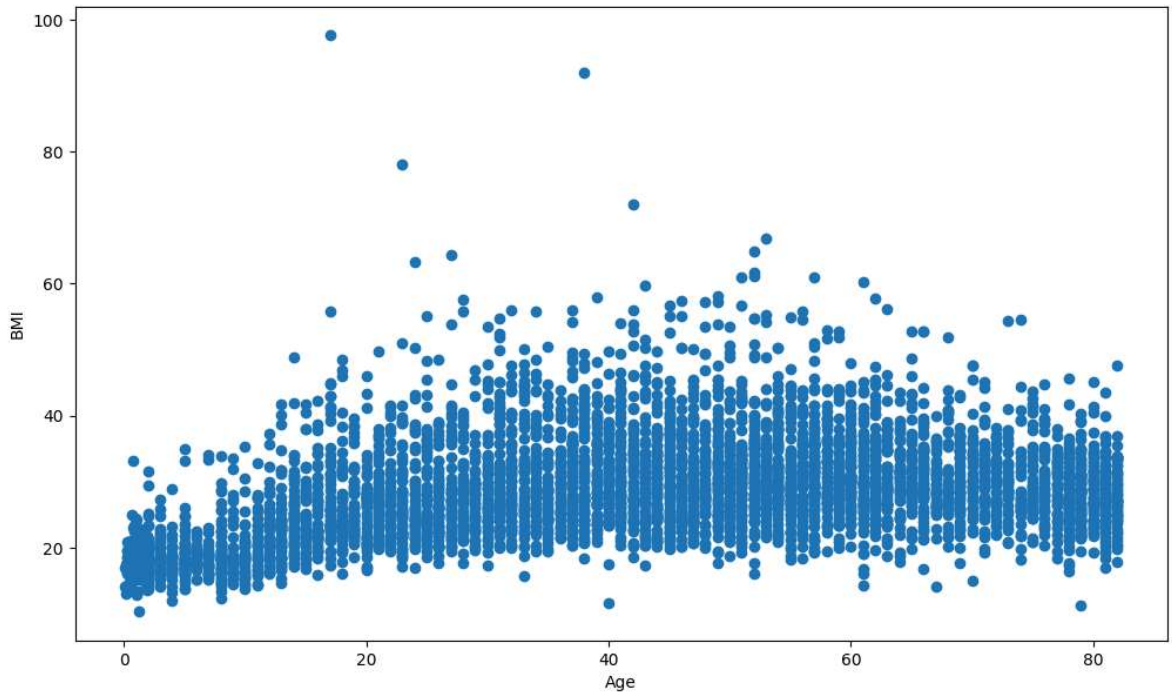
Stroke Prediction Analysis

In [51]:
```
stroke_count=df['stroke'].value_counts()
stroke_percent=df['stroke'].value_counts(normalize=True)*100
table7=pd.DataFrame({'Count':stroke_count,'Percentage %':stroke_percent})
table7
```

Out[51]:

stroke	Count	Percentage %
0	4700	95.742514
1	209	4.257486

In [52]:
```
plt.figure(figsize=(12,7))
plt.scatter(x='age',y='bmi',data=df)
plt.xlabel('Age')
plt.ylabel('BMI')
plt.show()
```

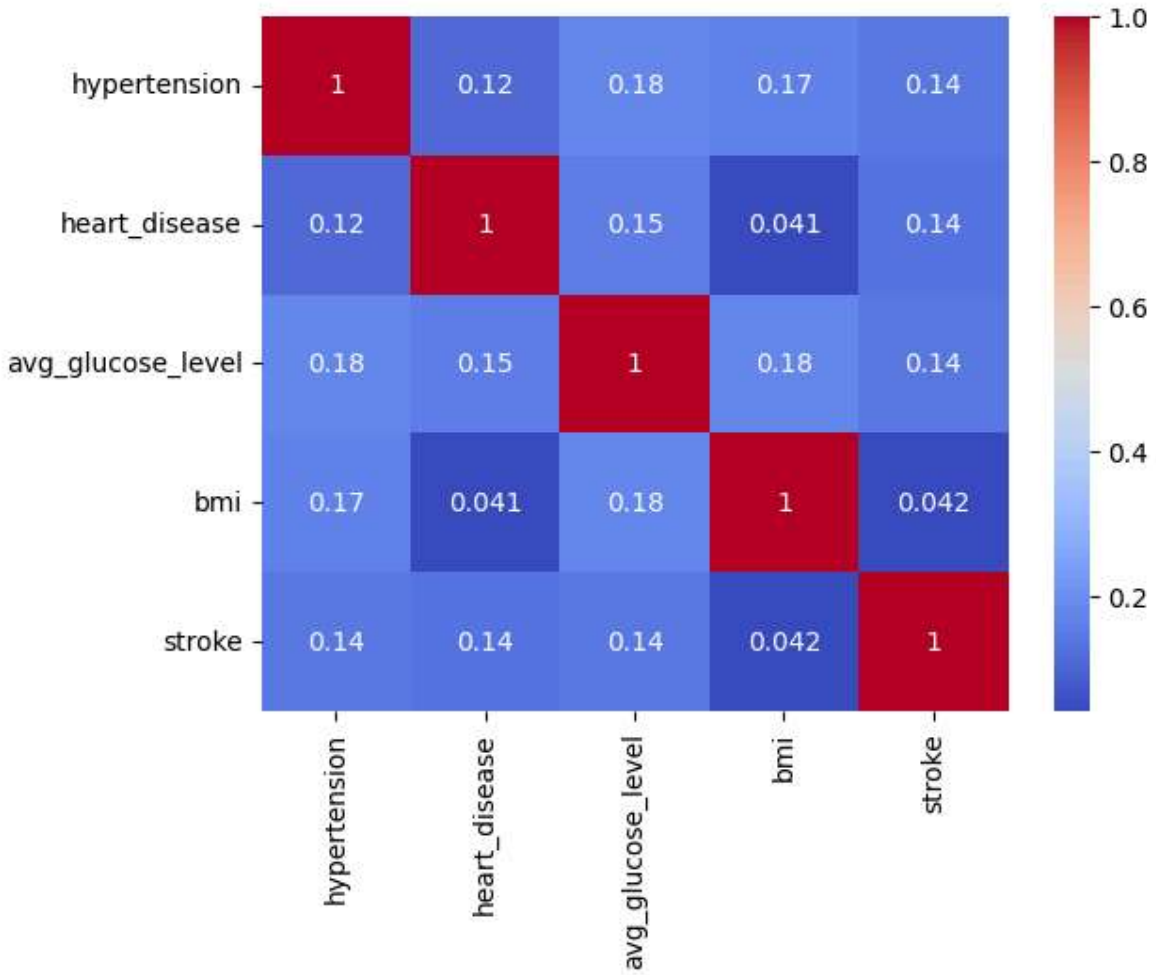


In [53]:
```
df.groupby(['Residence_type'])['stroke'].value_counts()
```

Out[53]:
```
Residence_type  stroke
Rural           0         2319
                1          100
Urban           0         2381
                1          109
Name: count, dtype: int64
```

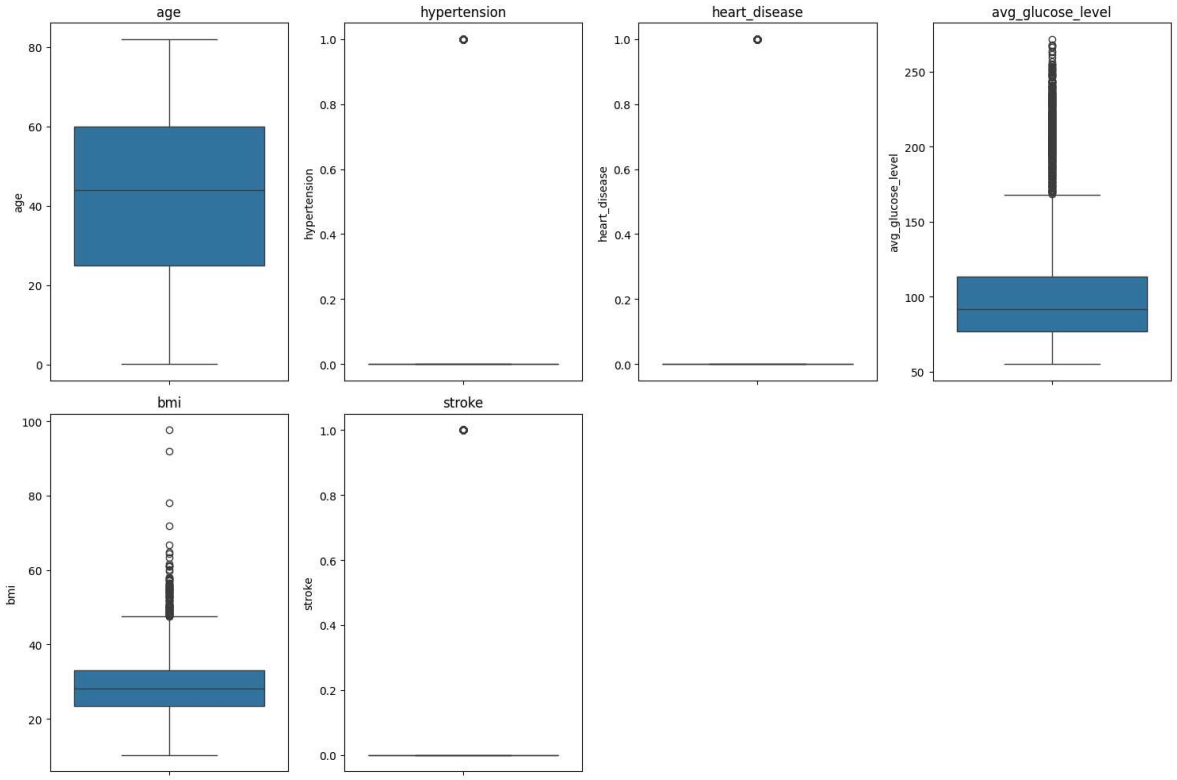
In [54]:
```
df2=df.iloc[:, 2:]
cm = df2.corr(numeric_only=True)
sns.heatmap(cm,annot=True,cmap='coolwarm')
```

Out[54]: <Axes: >

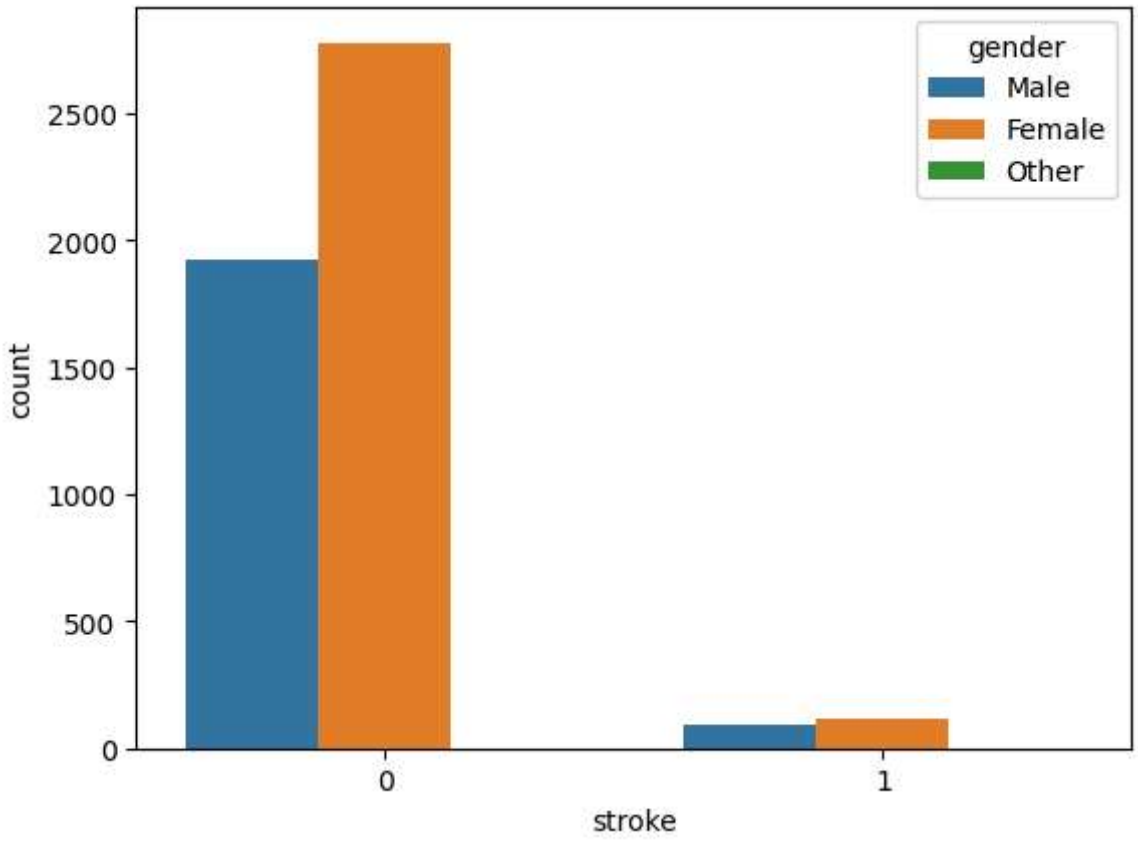


In [ ]:

In [55]:
```
numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns
n_cols = 4
n_rows = len(numerical_columns) // n_cols + 1

plt.figure(figsize=(15, 5 * n_rows))

# Plot each numerical column as a boxplot
for i, col_name in enumerate(numerical_columns, 1):
    plt.subplot(n_rows, n_cols, i)
    sns.boxplot(y=df[col_name])
    plt.title(col_name)

# Adjust the layout for better spacing
plt.tight_layout()

# Display the plot
plt.show()
```

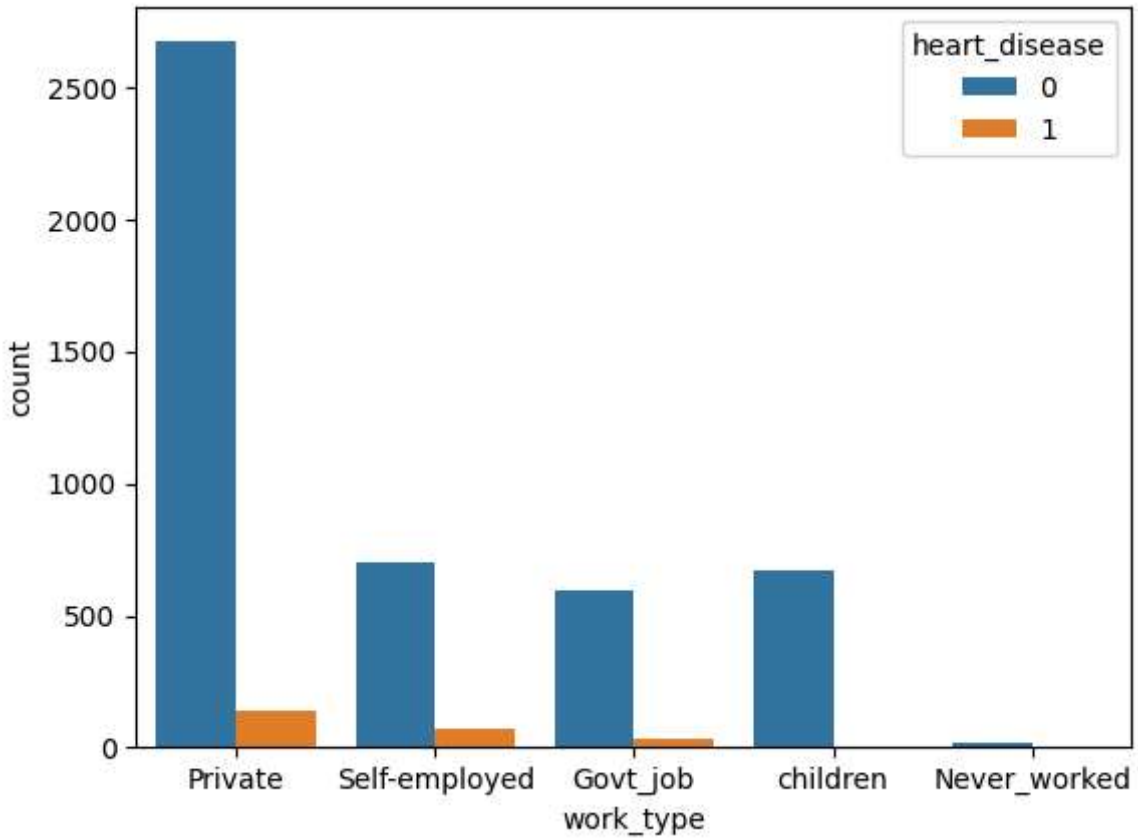


In [56]:
```
sns.countplot(x='stroke',hue='gender', data=df)
plt.show()
```



In [57]:
```
sns.countplot(x = 'work_type',hue='heart_disease' ,data=df)
```

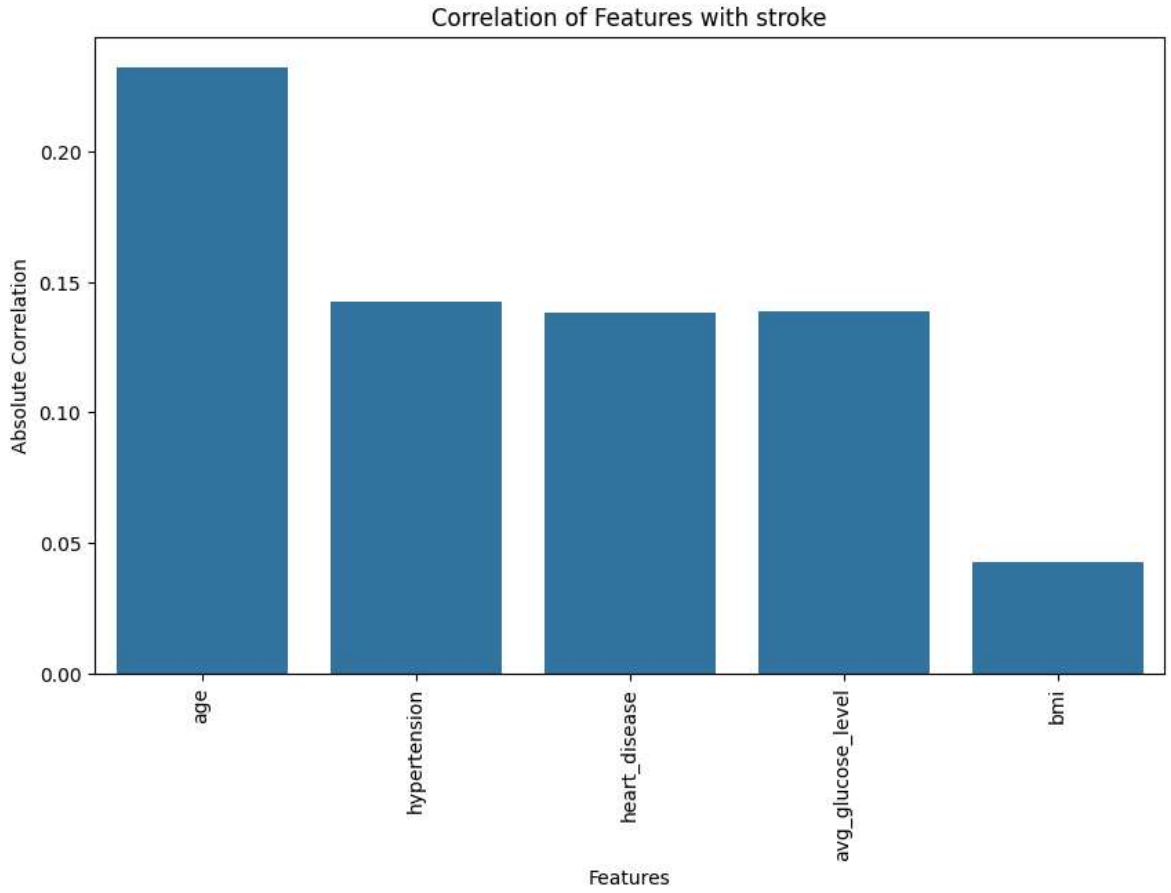
Out[57]:
```
<Axes: xlabel='work_type', ylabel='count'>
```



Correlations

In [59]:
```
# Select columns of type int or float
obj_cols = df.select_dtypes(include=['int64', 'float64']).columns

key = []
vals = []

# Iterate over numeric columns and compute correlation with 'RiskScore'
for n in obj_cols:
    if n=='stroke':
        continue
    key.append(n)
    vals.append(abs(df['stroke'].corr(df[n])))
    print(n,": ",df['stroke'].corr(df[n]))

# Plot the values
plt.figure(figsize=(10,6))
sns.barplot(x=key, y=vals)
plt.xticks(rotation=90)  # Rotate x-axis labels if necessary
plt.title('Correlation of Features with stroke')
plt.ylabel('Absolute Correlation')
plt.xlabel('Features')
plt.show()
```

```
age :  0.23233085553484958
hypertension :  0.1425146058811262
heart_disease :  0.13793778762219294
avg_glucose_level :  0.13893586200507035
bmi :  0.04237366114923355
```



In [60]:
```
health_data = pd.read_csv(r'C:\Users\Udayabhanu\Downloads\Stroke Prediction\S
```

In [61]:
```
health_data = health_data.drop('id',axis=1)
```

In [62]:
```
from sklearn.impute import SimpleImputer
impute= SimpleImputer(missing_values=np.nan,strategy='median')

health_data['bmi']=pd.DataFrame(impute.fit_transform(np.array(health_data['bm
health_data.isnull().sum()
```

Out[62]:
```
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 [63]:
```
from sklearn.preprocessing import OneHotEncoder
OH=OneHotEncoder()

for col in health_data.select_dtypes(include=['object']):
  health_data[col]=OH.fit_transform(health_data[[col]]).toarray()
health_data.head()
```

Out[63]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glu
0	0.0	67.0	0	1	0.0	0.0	0.0	
1	1.0	61.0	0	0	0.0	0.0	1.0	
2	0.0	80.0	0	1	0.0	0.0	1.0	
3	1.0	49.0	0	0	0.0	0.0	0.0	
4	1.0	79.0	1	0	0.0	0.0	1.0	

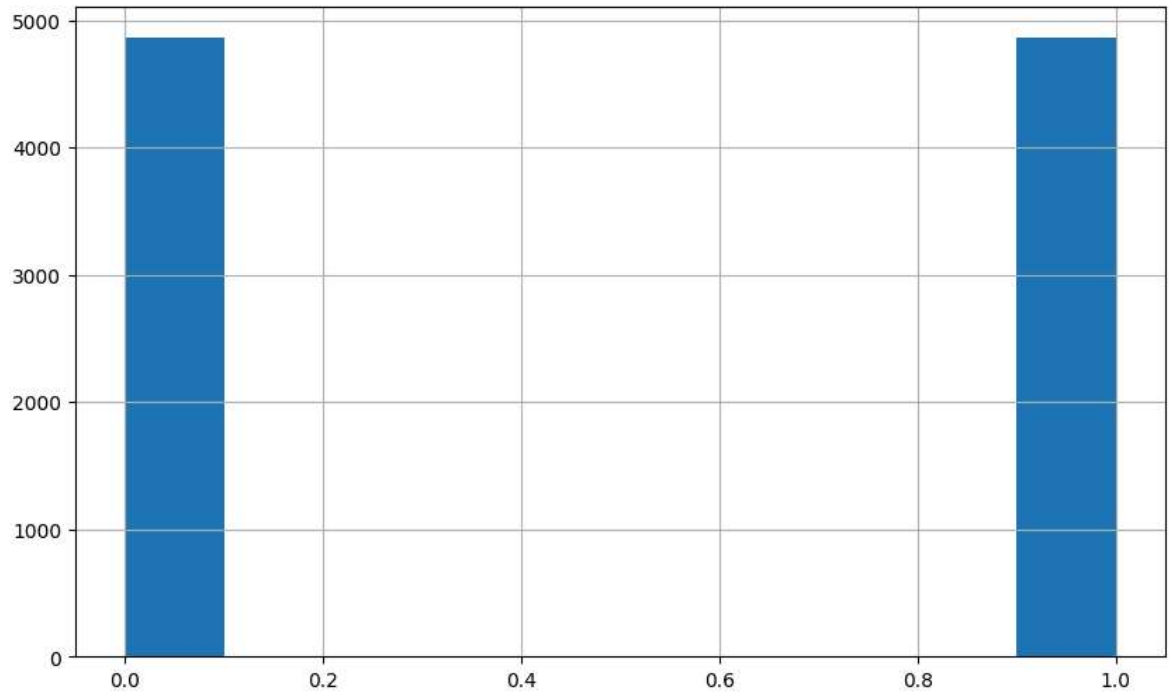
In [64]:
```
x=health_data.drop('stroke',axis=1)
y=health_data['stroke']
```

Handling unbalanced data

In [66]:
```
from imblearn.over_sampling import SMOTE
smote=SMOTE()

x_smote,y_smote=smote.fit_resample(x,y)
```

In [67]:
```
plt.figure(figsize=(10, 6))
y_smote.hist()
```

Out[67]: <Axes: >



Robust scaler

In [69]:
```
from sklearn.preprocessing import RobustScaler
rs=RobustScaler()

x_smote=rs.fit_transform(x_smote)
```

In [70]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(x_smote,y_smote,test_size=0.2,
```

Machine Learning

In [72]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall
from sklearn.linear_model import  LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

# Dictionary of models
models = {

    'LogisticRegression': LogisticRegression(max_iter = 2000),

    'RandomForestClassifier': RandomForestClassifier(),

    'KNeighborsClassifier' : KNeighborsClassifier(),

    'DecisionTreeClassifier': DecisionTreeClassifier(),

    'GaussianNB'            : GaussianNB(),

    'Support Vector Machine' : SVC() }
```

In [73]:
```
from tqdm import tqdm
# Fit models, predict and calculate accuracy and F1 score
results = []
models_name = []
for name, model in tqdm(models.items()):
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    f1 = f1_score(y_test, predictions, average='weighted')
    precision = precision_score(y_test, predictions, average='weighted')
    recall = recall_score(y_test, predictions, average='weighted')
    models_name.append(name)
    results.append([accuracy,precision,recall,f1])
```

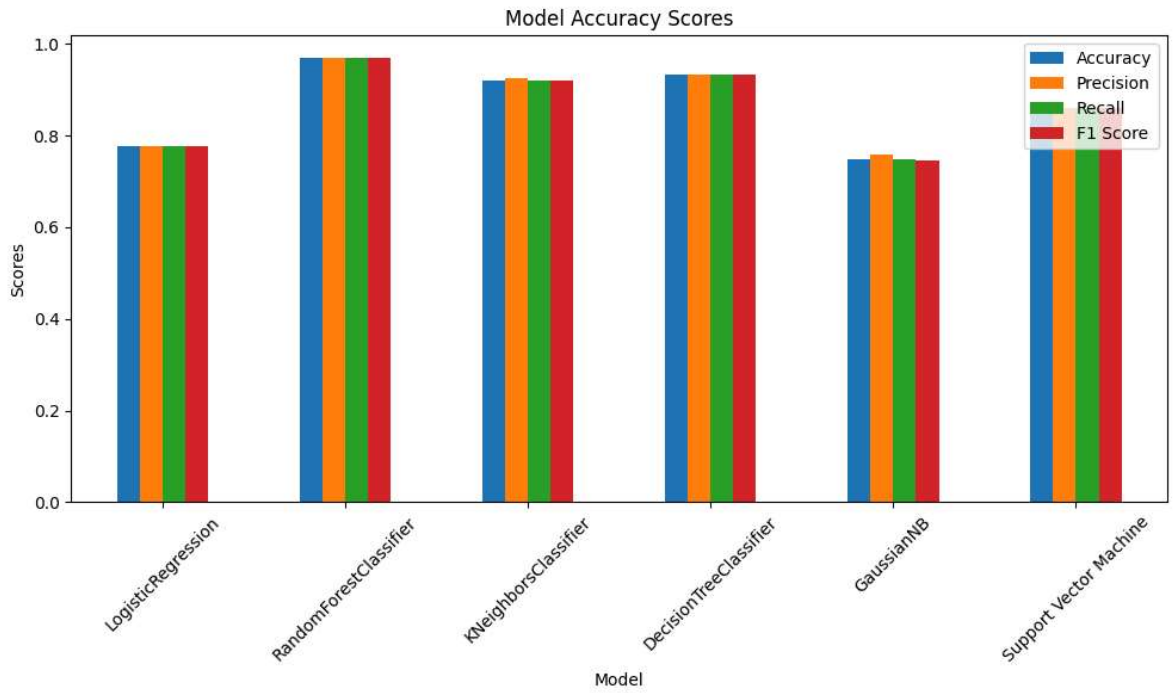
```
100%|██████████| 6/6 [00:02<00:00,  2.42it/s]
```

In [74]:
```
Model_accuracy = pd.DataFrame(results,index=models_name,columns = ['Accuracy'

Model_accuracy
```

Out[74]:

	Accuracy	Precision	Recall	F1 Score
LogisticRegression	0.776350	0.777269	0.776350	0.776187
RandomForestClassifier	0.969152	0.969596	0.969152	0.969143
KNeighborsClassifier	0.919280	0.924245	0.919280	0.919056
DecisionTreeClassifier	0.933676	0.934123	0.933676	0.933662
GaussianNB	0.749100	0.758677	0.749100	0.746848
Support Vector Machine	0.860154	0.860258	0.860154	0.860148

In [75]:
```
Model_accuracy.plot(kind='bar', figsize=(10, 6))

# Customizing the plot

plt.xlabel('Model')
plt.ylabel('Scores')
plt.title('Model Accuracy Scores')
plt.xticks(rotation=45)  # Rotate model names for better readability
plt.legend(loc='upper right')
plt.tight_layout()  # Adjust layout to fit labels

# Display the plot
plt.show()
```



Neural Network

In [77]:
```
import tensorflow as tf
from tensorflow import keras
```

In [78]:
```
model = keras.Sequential()
model.add(keras.layers.Dense(50,  activation="relu", input_shape = X_train.sh
model.add(keras.layers.Dense(40, activation="relu"))
model.add(keras.layers.Dense(30, activation="relu"))
model.add(keras.layers.Dense(20, activation="relu"))
model.add(keras.layers.Dense(10, activation="relu"))
model.add(keras.layers.Dense(1, activation="sigmoid"))
```

```
C:\Users\Udayabhanu\AppData\Roaming\Python\Python312\site-packages\keras\src
\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_d
im` argument to a layer. When using Sequential models, prefer using an `Inpu
t(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [79]:
```
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy

# Training and evaluating the model
history = model.fit(X_train, y_train, epochs=100, batch_size=100, validation_
```

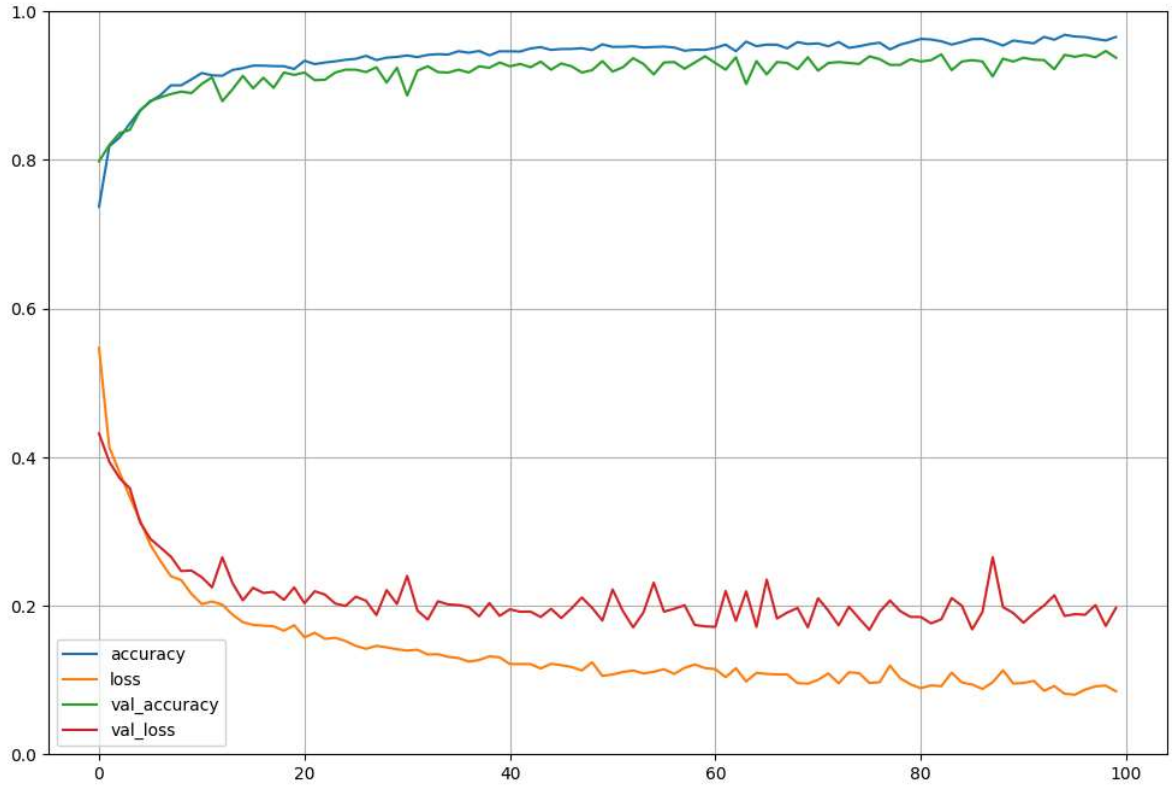
```
Epoch 1/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.6875 - loss: 0.6108
- val_accuracy: 0.7982 - val_loss: 0.4319
Epoch 2/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8061 - loss: 0.4293
- val_accuracy: 0.8201 - val_loss: 0.3938
Epoch 3/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8261 - loss: 0.3899
- val_accuracy: 0.8361 - val_loss: 0.3719
Epoch 4/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8471 - loss: 0.3492
- val_accuracy: 0.8406 - val_loss: 0.3582
Epoch 5/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8692 - loss: 0.3098
- val_accuracy: 0.8663 - val_loss: 0.3123
Epoch 6/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8741 - loss: 0.2887
- val_accuracy: 0.8798 - val_loss: 0.2901
Epoch 7/100
```

In [80]:
```
model.predict(X_test)
```

```
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

Out[80]:
```
array([[1.4419915e-07],
       [9.9999750e-01],
       [9.6922344e-01],
       ...,
       [9.9995506e-01],
       [2.5440505e-08],
       [9.9983770e-01]], dtype=float32)
```

In [81]:
```
import pandas as pd
import matplotlib.pyplot as plt
pd.DataFrame(history.history).plot(figsize=(12, 8))
plt.grid(True)
plt.gca().set_ylim(0, 1) # set the vertical range to [0-1]
plt.show()

# Evaluate the model
model_evaluate = model.evaluate(X_test, y_test)
print("Loss     : ",model_evaluate[0])
print("accuracy : ",model_evaluate[1])
```




```
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9223 - loss: 0.2063
Loss     :  0.22870728373527527
accuracy :  0.9244216084480286
```

Conclusion

The project involved analyzing health data to predict stroke risk. Key achievements include:

- Preprocessing data by handling missing values and encoding features.
- Conducting Exploratory Data Analysis (EDA) to visualize key health factors.
- Training machine learning models like logistic regression and random forests.
- Evaluating model performance and identifying important health indicators for stroke prediction.