In [1]:

```
import pandas as pd
import numpy as np

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

In [2]:

```
dataset = pd.read_csv(r'D:\harsha\Monty Corps Bangalore\Brain Stroke Project\healthcare-
```

In [3]:

```
dataset
```

Out[3]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence
0	9046	Male	67.0	0	1	Yes	Private	U
1	51676	Female	61.0	0	0	Yes	Self-employed	
2	31112	Male	80.0	0	1	Yes	Private	
3	60182	Female	49.0	0	0	Yes	Private	U
4	1665	Female	79.0	1	0	Yes	Self-employed	
...	...	...	...	...	...	...	...	
5105	18234	Female	80.0	1	0	Yes	Private	U
5106	44873	Female	81.0	0	0	Yes	Self-employed	U
5107	19723	Female	35.0	0	0	Yes	Self-employed	
5108	37544	Male	51.0	0	0	Yes	Private	
5109	44679	Female	44.0	0	0	Yes	Govt_job	U

5110 rows × 12 columns

In [4]:

```
dataset.head()
```

Out[4]:

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

In [5]:

```
dataset.shape
```

Out[5]:

```
(5110, 12)
```

In [6]:

```
dataset.columns
```

Out[6]:

```
Index(['id', 'gender', 'age', 'hypertension', 'heart_disease', 'ever_marri
ed',
       'work_type', 'Residence_type', 'avg_glucose_level', 'bmi',
       'smoking_status', 'stroke'],
      dtype='object')
```

In [7]:

```
dataset.describe()
```

Out[7]:

	id	age	hypertension	heart_disease	avg_glucose_level	bı
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	4909.0000(
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.8932
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.8540(
min	67.000000	0.080000	0.000000	0.000000	55.120000	10.3000(
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.5000(
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.1000(
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	33.1000(
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.6000(

In [8]:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 5110 non-null   int64  
 1   gender             5110 non-null   object 
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64  
 4   heart_disease      5110 non-null   int64  
 5   ever_married       5110 non-null   object 
 6   work_type          5110 non-null   object 
 7   Residence_type     5110 non-null   object 
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object 
 11  stroke             5110 non-null   int64  
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

In [9]:

```
dataset.isnull().sum()
```

Out[9]:

```
id                   0
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                201
smoking_status       0
stroke               0
dtype: int64
```

In [10]:

```
dataset['bmi'].dtype
```

Out[10]:

```
dtype('float64')
```

In [11]:

```
dataset['bmi']=dataset['bmi'].fillna(dataset['bmi'].mean())
```

In [12]:

```
data_distribution = dataset['stroke'].value_counts(normalize=True)
print(data_distribution)
```

```
0    0.951272
1    0.048728
Name: stroke, dtype: float64
```

In [13]:

```
dataset.isnull().sum()
```

Out[13]:

```
id                 0
gender             0
age                0
hypertension       0
heart_disease      0
ever_married       0
work_type          0
Residence_type     0
avg_glucose_level  0
bmi                0
smoking_status     0
stroke             0
dtype: int64
```

In [14]:

```
dataset.head()
```

Out[14]:

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

In [15]:

```
dataset['work_type'].groupby(dataset['work_type']).size()
```

Out[15]:

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

In [16]:

```
dataset['age'] = dataset['age'].astype(int)
```

In [17]:

```
dataset
```

Out[17]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_
0	9046	Male	67	0	1	Yes	Private	U
1	51676	Female	61	0	0	Yes	Self-employed	
2	31112	Male	80	0	1	Yes	Private	
3	60182	Female	49	0	0	Yes	Private	U
4	1665	Female	79	1	0	Yes	Self-employed	
...	...	...	...	...	...	...	...	
5105	18234	Female	80	1	0	Yes	Private	U
5106	44873	Female	81	0	0	Yes	Self-employed	U
5107	19723	Female	35	0	0	Yes	Self-employed	
5108	37544	Male	51	0	0	Yes	Private	
5109	44679	Female	44	0	0	Yes	Govt_job	U

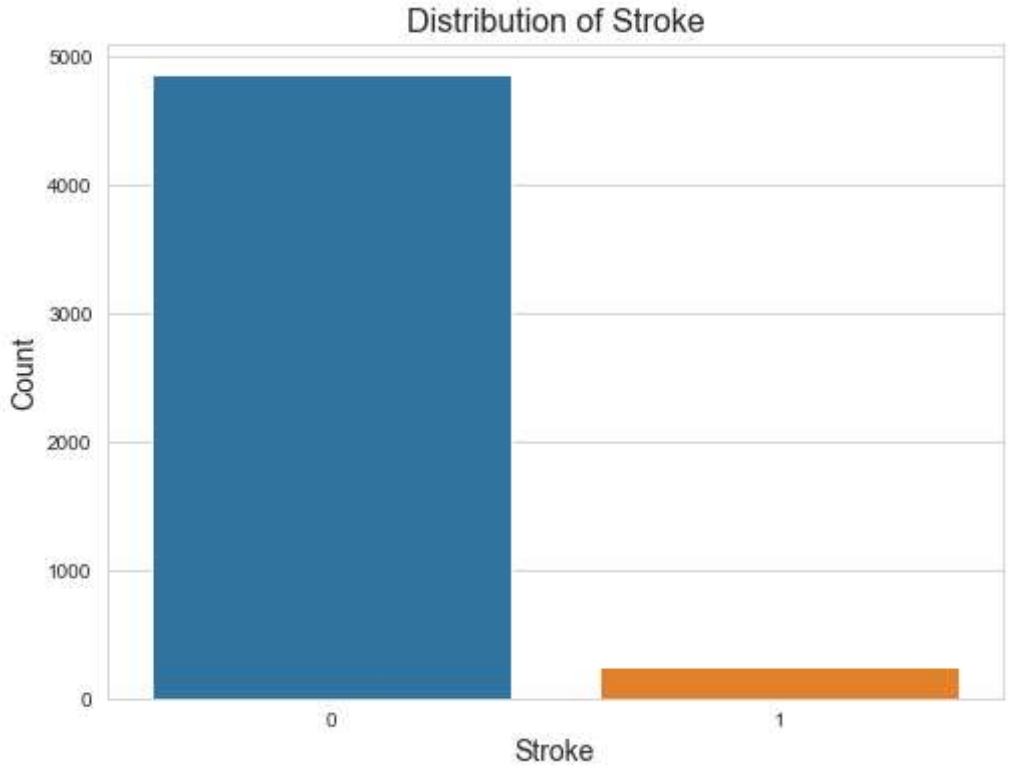
5110 rows × 12 columns

In [18]:

```
# Importing Visualisation Library
import seaborn as sns
import matplotlib.pyplot as plt
```

In [19]:

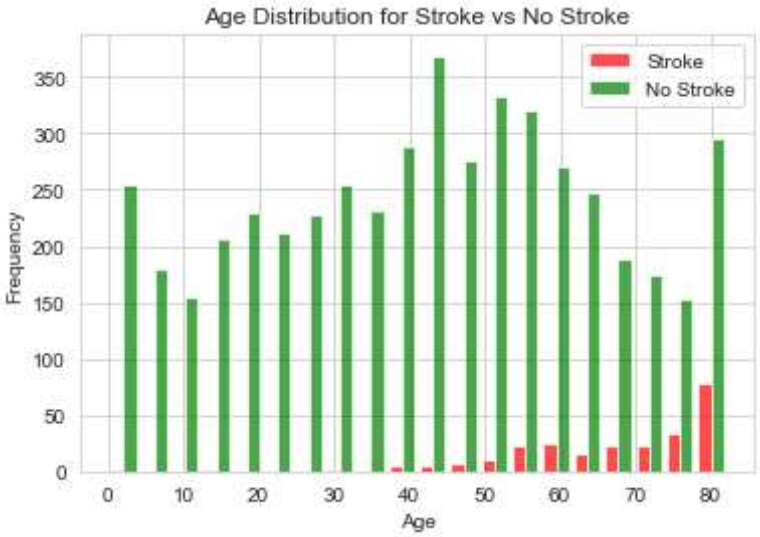
```
sns.set_style('whitegrid')
plt.figure(figsize=(8, 6))
sns.countplot(x='stroke', data=dataset)
plt.xlabel('Stroke', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.title('Distribution of Stroke', fontsize=16)
plt.show()
```



In [20]:

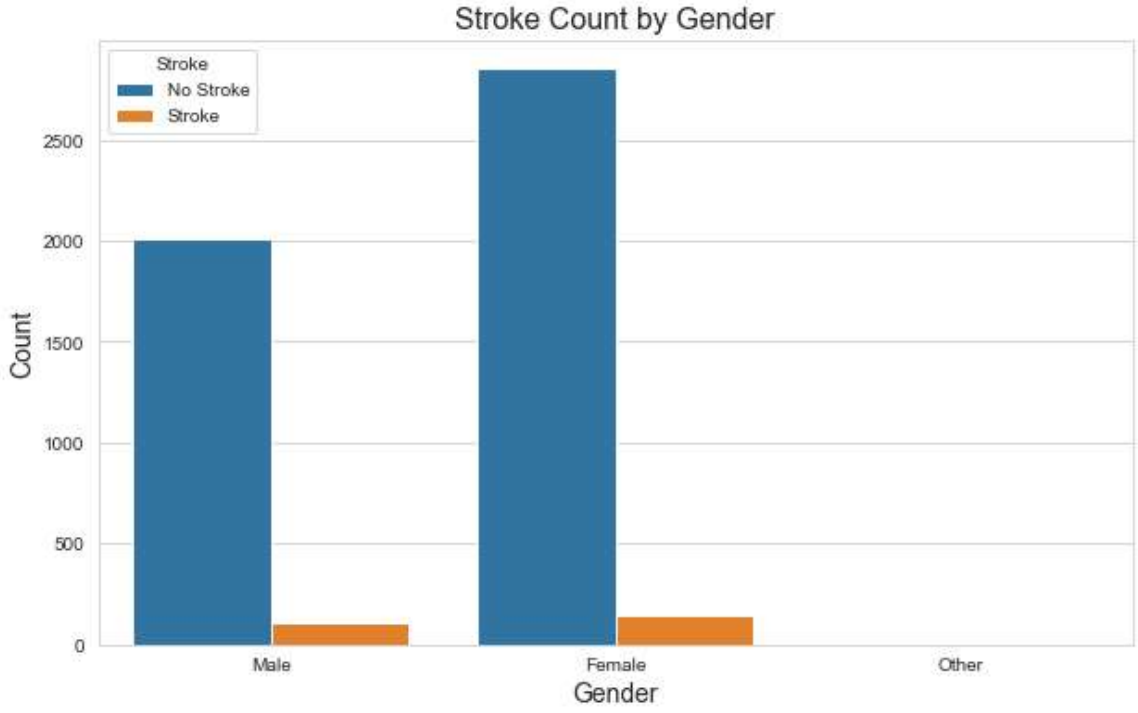
```
# Separate data into stroke and no-stroke groups
stroke_group = dataset[dataset['stroke'] == 1]
no_stroke_group = dataset[dataset['stroke'] == 0]

# Create a histogram for age distribution
plt.hist([stroke_group['age'], no_stroke_group['age']], bins=20, color=['red', 'green'],
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Age Distribution for Stroke vs No Stroke')
plt.legend()
plt.show()
```



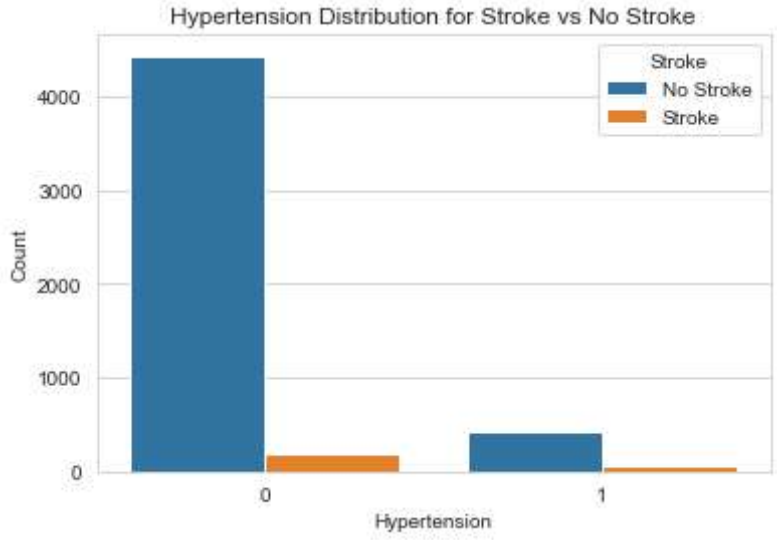
In [21]:

```
plt.figure(figsize=(10, 6))
sns.countplot(x='gender', hue='stroke', data=dataset)
plt.xlabel('Gender', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.title('Stroke Count by Gender', fontsize=16)
plt.legend(title='Stroke', labels=['No Stroke', 'Stroke'])
plt.show()
```



In [22]:

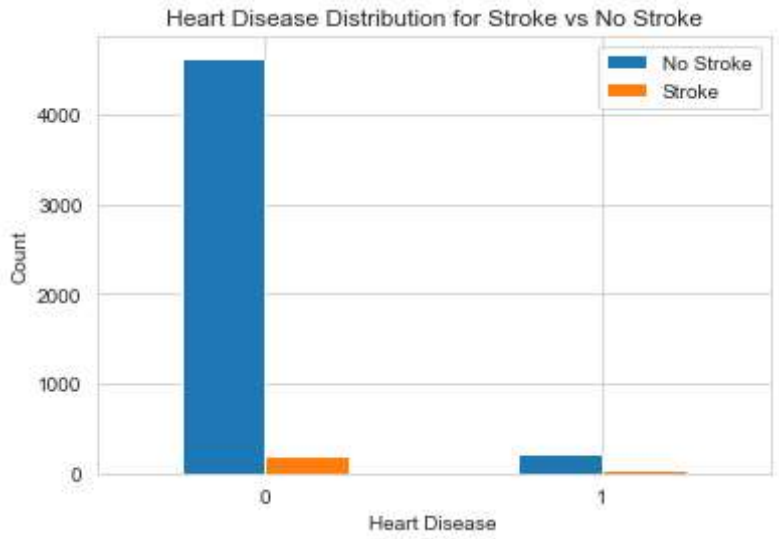
```
# Create a count plot
sns.countplot(data=dataset, x='hypertension', hue='stroke')
plt.xlabel('Hypertension')
plt.ylabel('Count')
plt.title('Hypertension Distribution for Stroke vs No Stroke')
plt.legend(title='Stroke', labels=['No Stroke', 'Stroke'])
plt.show()
```



In [23]:

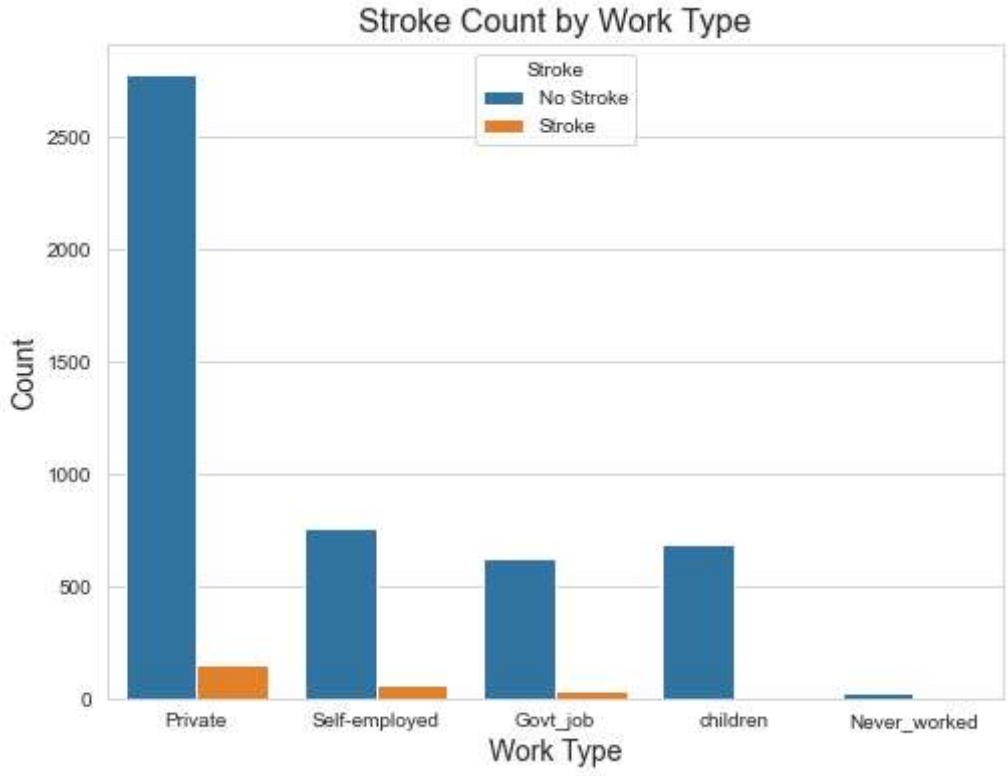
```
# Count the occurrences of stroke for each heart disease status
heart_disease_stroke_counts = dataset.groupby(['heart_disease', 'stroke']).size().unstac

# Create a bar chart
heart_disease_stroke_counts.plot(kind='bar')
plt.xlabel('Heart Disease')
plt.ylabel('Count')
plt.title('Heart Disease Distribution for Stroke vs No Stroke')
plt.xticks(rotation=0)
plt.legend(['No Stroke', 'Stroke'])
plt.show()
```



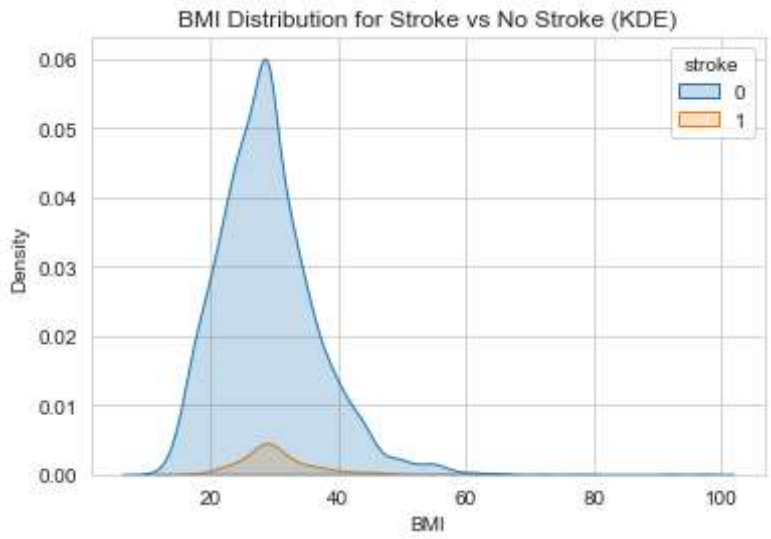
In [24]:

```
plt.figure(figsize=(8, 6))
sns.countplot(x='work_type', hue='stroke', data=dataset)
plt.xlabel('Work Type', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.title('Stroke Count by Work Type', fontsize=16)
plt.legend(title='Stroke', labels=['No Stroke', 'Stroke'])
plt.show()
```



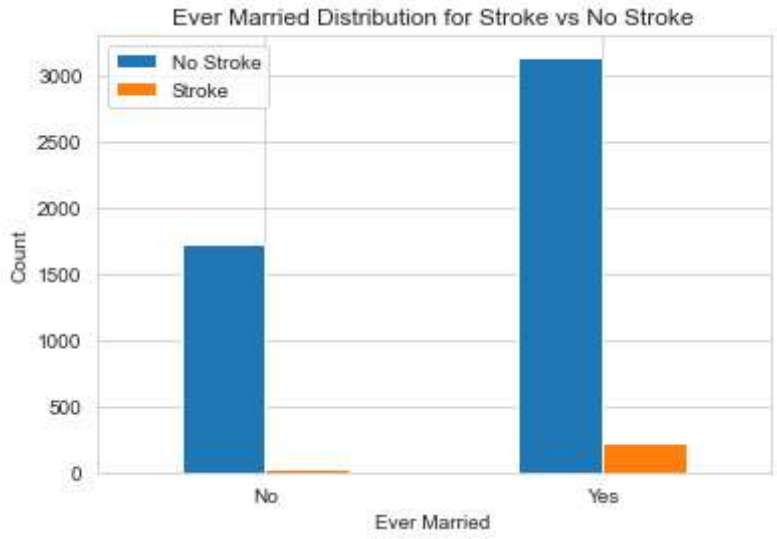
In [25]:

```
# Create a KDE plot
sns.kdeplot(data=dataset, x='bmi', hue='stroke', fill=True)
plt.xlabel('BMI')
plt.ylabel('Density')
plt.title('BMI Distribution for Stroke vs No Stroke (KDE)')
plt.show()
```



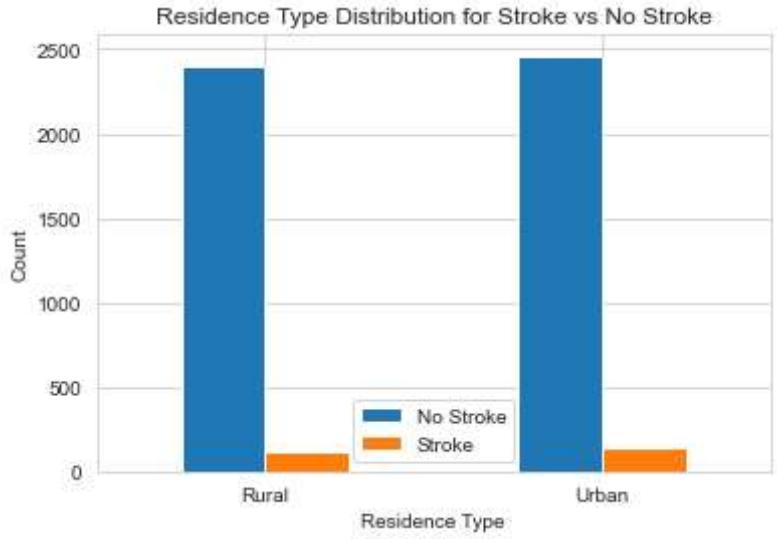
In [26]:

```
# Create a bar chart
ever_married_stroke_counts = dataset.groupby(['ever_married', 'stroke']).size().unstack(
ever_married_stroke_counts.plot(kind='bar')
plt.xlabel('Ever Married')
plt.ylabel('Count')
plt.title('Ever Married Distribution for Stroke vs No Stroke')
plt.xticks(rotation=0)
plt.legend(['No Stroke', 'Stroke'])
plt.show()
```



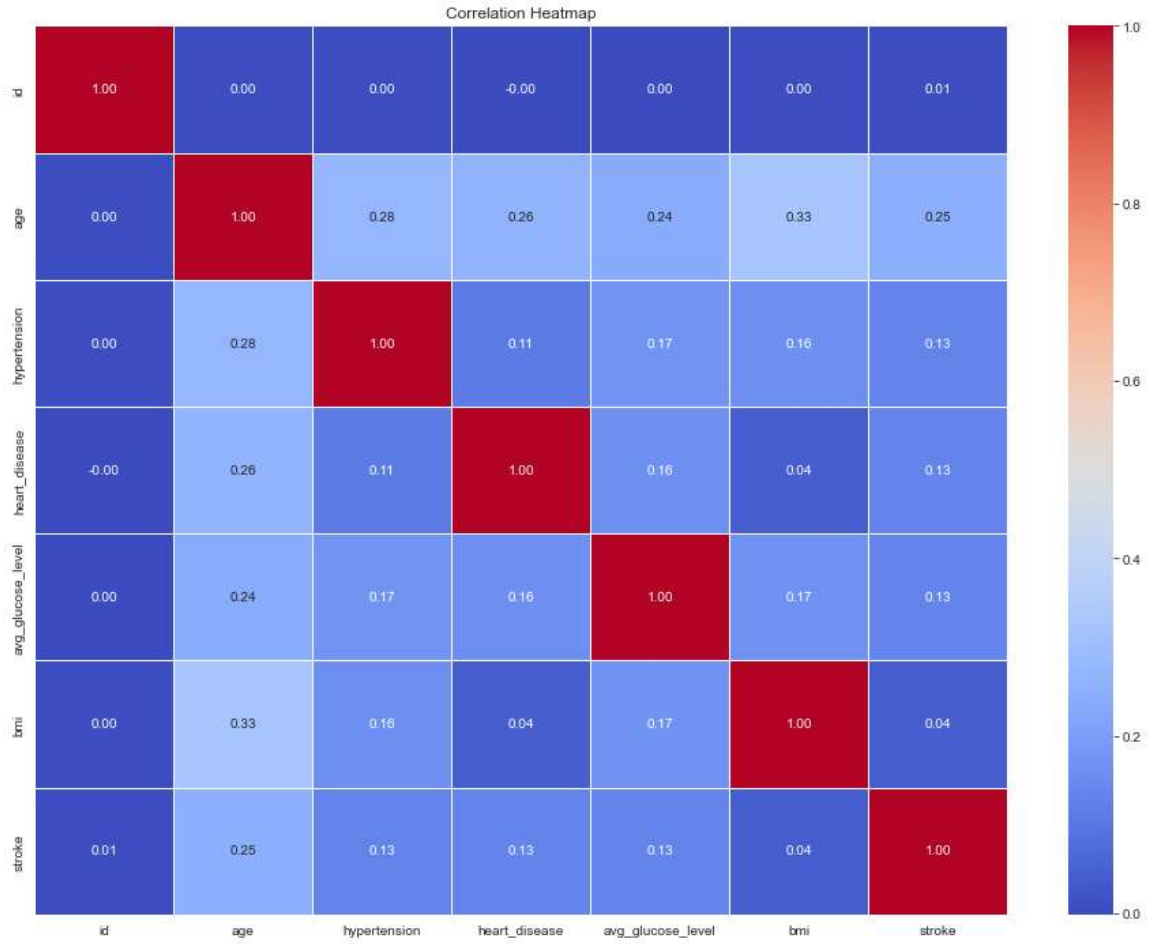
In [27]:

```
# Create a bar chart
residence_type_stroke_counts = dataset.groupby(['Residence_type', 'stroke']).size().unst
residence_type_stroke_counts.plot(kind='bar')
plt.xlabel('Residence Type')
plt.ylabel('Count')
plt.title('Residence Type Distribution for Stroke vs No Stroke')
plt.xticks(rotation=0)
plt.legend(['No Stroke', 'Stroke'])
plt.show()
```



In [28]:

```
#df_balanced = pd.concat([pd.DataFrame(X_train_balanced, columns=X.columns), pd.Series(y_
plt.figure(figsize=(16,12))
sns.heatmap(dataset.corr(), annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



In [29]:

```
from sklearn.preprocessing import LabelEncoder
le= LabelEncoder()
```

In [30]:

```
dataset['gender'] = le.fit_transform(dataset['gender'])
dataset['ever_married'] = le.fit_transform(dataset['ever_married'])
dataset['work_type'] = le.fit_transform(dataset['work_type'])
dataset['Residence_type'] = le.fit_transform(dataset['Residence_type'])
dataset['smoking_status'] = le.fit_transform(dataset['smoking_status'])
```

In [31]:

```
dataset.head()
```

Out[31]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_typ
0	9046	1	67	0	1	1	2	
1	51676	0	61	0	0	1	3	
2	31112	1	80	0	1	1	2	
3	60182	0	49	0	0	1	2	
4	1665	0	79	1	0	1	3	

In [32]:

```
dataset['id'].unique
```

Out[32]:

```
<bound method Series.unique of 0        9046
1       51676
2       31112
3       60182
4        1665
        ...
5105    18234
5106    44873
5107    19723
5108    37544
5109    44679
Name: id, Length: 5110, dtype: int64>
```

In [33]:

```
dataset['id'].groupby(dataset['id']).size()
```

Out[33]:

```
id
67       1
77       1
84       1
91       1
99       1
        ..
72911    1
72914    1
72915    1
72918    1
72940    1
Name: id, Length: 5110, dtype: int64
```

In [34]:

```
X=dataset.iloc[:,0:11]
X
```

Out[34]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_
0	9046	1	67	0	1	1	2	
1	51676	0	61	0	0	1	3	
2	31112	1	80	0	1	1	2	
3	60182	0	49	0	0	1	2	
4	1665	0	79	1	0	1	3	
...	...	...	...	...	...	...	...	
5105	18234	0	80	1	0	1	2	
5106	44873	0	81	0	0	1	3	
5107	19723	0	35	0	0	1	3	
5108	37544	1	51	0	0	1	2	
5109	44679	0	44	0	0	1	0	

5110 rows × 11 columns

In [35]:

```
y= dataset.iloc[:,-1]
y
```

Out[35]:

```
0       1
1       1
2       1
3       1
4       1
       ..
5105    0
5106    0
5107    0
5108    0
5109    0
Name: stroke, Length: 5110, dtype: int64
```

In [36]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train,y_test= train_test_split(X,y,test_size=0.2, random_state=0)
```

In [37]:

```
# Feature Scaling

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Train the model using Logistic Regression

In [38]:

```
#Fitting Logistic Regression to the training set

from sklearn.linear_model import LogisticRegression
classifier_lr= LogisticRegression(solver= 'liblinear', penalty= 'l2', C= 10,random_state
classifier_lr.fit(X_train, y_train)
```

Out[38]:

```
LogisticRegression(C=10, random_state=0, solver='liblinear')
```

In [39]:

```
y_pred_lr = classifier_lr.predict(X_test)
print('y_pred for test',y_pred_lr)
print('\n')

y_pred_lr_train = classifier_lr.predict(X_train)
print('y_pred for train ',y_pred_lr_train)
print('\n')

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm_lr = confusion_matrix(y_test, y_pred_lr)
print('Confusion Matrix for test \n',cm_lr)
print('\n')


from sklearn.metrics import confusion_matrix
cm_lr_train = confusion_matrix(y_train, y_pred_lr_train)
print('Confusion Matrix for train \n',cm_lr_train)
print('\n\n')


# This is to get the Models Accuracy
from sklearn.metrics import accuracy_score
ac_lr = accuracy_score(y_test, y_pred_lr)
print('Accuracy score for test ',ac_lr)
print('\n\n')

from sklearn.metrics import accuracy_score
ac_lr_ = accuracy_score(y_train, y_pred_lr_train)
print('Accuracy score for train ',ac_lr_)
print('\n\n')

bias_lr = classifier_lr.score(X_train,y_train)
print('Bias = ',bias_lr)
print('\n\n')

variance_lr = classifier_lr.score(X_test,y_test)
print('Variance = ',variance_lr)
print('\n\n')

# This is to get the Classification Report
from sklearn.metrics import classification_report
cr_lr = classification_report(y_test, y_pred_lr)
print('Classification report ', cr_lr)
```

```
y_pred for test [0 0 0 ... 0 0 0]


y_pred for train  [0 0 0 ... 0 0 0]


Confusion Matrix for test
 [[968   0]
 [ 54   0]]


Confusion Matrix for train
 [[3893    0]
 [ 195    0]]



Accuracy score for test  0.9471624266144814



Accuracy score for train  0.9522994129158513



Bias =  0.9522994129158513



Variance =  0.9471624266144814



Classification report                precision    recall  f1-score   suppo
rt

           0       0.95      1.00      0.97       968
           1       0.00      0.00      0.00        54

    accuracy                           0.95      1022
   macro avg       0.47      0.50      0.49      1022
weighted avg       0.90      0.95      0.92      1022
```

Train the model using SVM Algorithm

In [40]:

```
# Training the SVM model on the Training set

from sklearn.svm import SVC
svm_classifier = SVC(kernel= 'poly', degree= 3, C=1.0)
svm_classifier.fit(X_train, y_train)
```

Out[40]:

```
SVC(kernel='poly')
```

In [41]:

```
y_pred_svm = svm_classifier.predict(X_test)
y_pred_svm
```

Out[41]:

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [42]:

```
y_pred_svm_train = svm_classifier.predict(X_train)
y_pred_svm_train
```

Out[42]:

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [43]:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm_svm = confusion_matrix(y_test, y_pred_svm)
print(cm_svm)
```

```
[[966   2]
 [ 53   1]]
```

In [44]:

```
# This is to get the Models Accuracy
from sklearn.metrics import accuracy_score
ac_svm = accuracy_score(y_test, y_pred_svm)
print(ac_svm)
```

```
0.9461839530332681
```

In [45]:

```
from sklearn.metrics import accuracy_score
ac_svm_ = accuracy_score(y_train, y_pred_svm_train)
print(ac_svm_)
```

```
0.9527886497064579
```

In [46]:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm_svm = confusion_matrix(y_train, y_pred_svm_train)
print(cm_svm)
```

```
[[3888    5]
 [ 188    7]]
```

In [47]:

```
bias_svm = svm_classifier.score(X_train,y_train)
bias_svm
```

Out[47]:

```
0.9527886497064579
```

In [48]:

```
variance_svm = svm_classifier.score(X_test,y_test)
variance_svm
```

Out[48]:

```
0.9461839530332681
```

In [49]:

```
# This is to get the Classification Report
from sklearn.metrics import classification_report
cr_svm = classification_report(y_test, y_pred_svm)
cr_svm
```

Out[49]:

```
'              precision    recall  f1-score   support\n\n           0
0.95      1.00      0.97       968\n           1       0.33      0.02
0.04        54\n\n    accuracy                           0.95      1022\n
macro avg       0.64      0.51      0.50      1022\nweighted avg       0.9
2      0.95      0.92      1022\n'
```

Train model using KNN Algorithm

In [50]:

```
from sklearn.neighbors import KNeighborsClassifier
classifier_knn = KNeighborsClassifier(weights= 'uniform',
p= 2,
n_neighbors= 5,
leaf_size=30,
metric='minkowski',
algorithm= 'auto')
classifier_knn.fit(X_train,y_train)
```

Out[50]:

```
KNeighborsClassifier
KNeighborsClassifier()
```

In [51]:

```
y_pred_knn = classifier_knn.predict(X_test)
print('y_pred for test',y_pred_knn)
print('\n\n')

y_pred_knn_train = classifier_knn.predict(X_train)
print('y_pred for train ',y_pred_knn_train)
print('\n\n')

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm_knn = confusion_matrix(y_test, y_pred_knn)
print('Confusion Matrix for test \n',cm_knn)
print('\n\n')


from sklearn.metrics import confusion_matrix
cm_knn_train = confusion_matrix(y_train, y_pred_knn_train)
print('Confusion Matrix for train \n',cm_knn_train)
print('\n\n')


# This is to get the Models Accuracy
from sklearn.metrics import accuracy_score
ac_knn = accuracy_score(y_test, y_pred_knn)
print('Accuracy score for test ',ac_knn)
print('\n\n')

from sklearn.metrics import accuracy_score
ac_knn_ = accuracy_score(y_train, y_pred_knn_train)
print('Accuracy score for train ',ac_knn_)
print('\n\n')

bias_knn = classifier_knn.score(X_train,y_train)
print('Bias = ',bias_knn)
print('\n\n')

variance_knn = classifier_knn.score(X_test,y_test)
print('Variance = ',variance_knn)
print('\n\n')

# This is to get the Classification Report
from sklearn.metrics import classification_report
cr_knn = classification_report(y_test, y_pred_knn)
print('Classification report ', cr_knn)
```

```
y_pred for test [0 0 0 ... 0 0 0]



y_pred for train  [0 0 0 ... 0 0 0]



Confusion Matrix for test
 [[967   1]
 [ 54   0]]



Confusion Matrix for train
 [[3889    4]
 [ 188    7]]



Accuracy score for test  0.9461839530332681



Accuracy score for train  0.9530332681017613



Bias =  0.9530332681017613



Variance =  0.9461839530332681



Classification report                precision    recall  f1-score   suppo
rt

           0       0.95      1.00      0.97       968
           1       0.00      0.00      0.00        54

    accuracy                           0.95      1022
   macro avg       0.47      0.50      0.49      1022
weighted avg       0.90      0.95      0.92      1022
```

Train the model using Decision Tree Algorithm

In [52]:

```
from sklearn.tree import DecisionTreeClassifier
classifier_dt = DecisionTreeClassifier(splitter = 'best',
 min_samples_split= 19,
 min_samples_leaf= 32,
 max_depth= 14,
 criterion='log_loss')
classifier_dt.fit(X_train, y_train)
```

Out[52]:

```
DecisionTreeClassifier(criterion='log_loss', max_depth=14, min_samples_le
af=32,
                       min_samples_split=19)
```

In [53]:

```
y_pred_dt = classifier_dt.predict(X_test)
print('y_pred for test',y_pred_dt)
print('\n')

y_pred_dt_train = classifier_dt.predict(X_train)
print('y_pred for train ',y_pred_dt_train)
print('\n')

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm_dt = confusion_matrix(y_test, y_pred_dt)
print('Confusion Matrix for test \n',cm_dt)
print('\n')


from sklearn.metrics import confusion_matrix
cm_dt_train = confusion_matrix(y_train, y_pred_dt_train)
print('Confusion Matrix for train \n',cm_dt_train)
print('\n')


# This is to get the Models Accuracy
from sklearn.metrics import accuracy_score
ac_dt = accuracy_score(y_test, y_pred_dt)
print('Accuracy score for test ',ac_dt)
print('\n')

from sklearn.metrics import accuracy_score
ac_dt_ = accuracy_score(y_train, y_pred_dt_train)
print('Accuracy score for train ',ac_dt_)
print('\n\n')

bias_dt = classifier_dt.score(X_train,y_train)
print('Bias = ',bias_dt)
print('\n\n')

variance_dt = classifier_dt.score(X_test,y_test)
print('Variance = ',variance_dt)
print('\n\n')

# This is to get the Classification Report
from sklearn.metrics import classification_report
cr_dt = classification_report(y_test, y_pred_dt)
print('Classification report ', cr_dt)
```

```
y_pred for test [0 0 0 ... 0 0 0]


y_pred for train  [0 0 0 ... 0 0 0]


Confusion Matrix for test
 [[960   8]
 [ 52   2]]


Confusion Matrix for train
 [[3877   16]
 [ 178   17]]


Accuracy score for test  0.9412915851272016


Accuracy score for train  0.9525440313111546



Bias =  0.9525440313111546



Variance =  0.9412915851272016



Classification report                precision    recall  f1-score   suppo
rt

           0       0.95      0.99      0.97       968
           1       0.20      0.04      0.06        54

    accuracy                           0.94      1022
   macro avg       0.57      0.51      0.52      1022
weighted avg       0.91      0.94      0.92      1022
```

Train the model using Random Forest Classifier

In [54]:

```
from sklearn.ensemble import RandomForestClassifier
classifier_rf = RandomForestClassifier(n_estimators= 995,
 min_samples_split= 41,
 min_samples_leaf= 38,
 max_depth= 20,
 criterion= 'gini')
classifier_rf.fit(X_train, y_train)
```

Out[54]:

```
RandomForestClassifier(max_depth=20, min_samples_leaf=38, min_samples_split=41,
                       n_estimators=995)
```

In [55]:

```
y_pred_rf = classifier_rf.predict(X_test)
print('y_pred for test',y_pred_rf)
print('\n')

y_pred_rf_train = classifier_rf.predict(X_train)
print('y_pred for train ',y_pred_rf_train)
print('\n')

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm_rf = confusion_matrix(y_test, y_pred_rf)
print('Confusion Matrix for test \n',cm_rf)
print('\n')


from sklearn.metrics import confusion_matrix
cm_rf_train = confusion_matrix(y_train, y_pred_rf_train)
print('Confusion Matrix for train \n',cm_rf_train)
print('\n')


# This is to get the Models Accuracy
from sklearn.metrics import accuracy_score
ac_rf = accuracy_score(y_test, y_pred_rf)
print('Accuracy score for test ',ac_rf)
print('\n')

from sklearn.metrics import accuracy_score
ac_rf_ = accuracy_score(y_train, y_pred_rf_train)
print('Accuracy score for train ',ac_rf_)
print('\n\n')

bias_rf = classifier_rf.score(X_train,y_train)
print('Bias = ',bias_rf)
print('\n\n')

variance_rf = classifier_rf.score(X_test,y_test)
print('Variance = ',variance_rf)
print('\n\n')

# This is to get the Classification Report
from sklearn.metrics import classification_report
cr_rf = classification_report(y_test, y_pred_rf)
print('Classification report ', cr_rf)
```

```
y_pred for test [0 0 0 ... 0 0 0]


y_pred for train  [0 0 0 ... 0 0 0]


Confusion Matrix for test
 [[968   0]
 [ 54   0]]


Confusion Matrix for train
 [[3893    0]
 [ 195    0]]


Accuracy score for test  0.9471624266144814


Accuracy score for train  0.9522994129158513



Bias =  0.9522994129158513



Variance =  0.9471624266144814



Classification report                precision    recall  f1-score   suppo
rt

           0       0.95      1.00      0.97       968
           1       0.00      0.00      0.00        54

    accuracy                           0.95      1022
   macro avg       0.47      0.50      0.49      1022
weighted avg       0.90      0.95      0.92      1022
```

In [ ]: