Heart Stroke Prediction

Importing libraries

In [ ]:
```
#Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import f1_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import log_loss
```

In [ ]:
```
#Loading the dataset
df = pd.read_csv('healthcare-dataset-stroke-data.csv')
df.head()
```

Out[ ]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residen
0	9046	Male	67.0	0	1	Yes	Private	
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	
4	1665	Female	79.0	1	0	Yes	Self-employed	

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

Data Preprocessing

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

Out[ ]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	s
count	5110.000000	5110.000000	5110.000000	5110.000000	4909.000000	5110.0(
mean	43.226614	0.097456	0.054012	106.147677	28.893237	0.0
std	22.612647	0.296607	0.226063	45.283560	7.854067	0.2
min	0.080000	0.000000	0.000000	55.120000	10.300000	0.0(
25%	25.000000	0.000000	0.000000	77.245000	23.500000	0.0(
50%	45.000000	0.000000	0.000000	91.885000	28.100000	0.0(
75%	61.000000	0.000000	0.000000	114.090000	33.100000	0.0(
max	82.000000	1.000000	1.000000	271.740000	97.600000	1.0(

In [ ]:
```
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 [ ]:
```
df['age'].astype(int)
```

Out[ ]:
```
0       67
1       61
2       80
3       49
4       79
        ..
5105    80
5106    81
5107    35
5108    51
5109    44
Name: age, Length: 5110, dtype: int32
```

In [ ]:
```
#Checking for null values
df.isnull().sum()
```

Out[ ]:
```
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 [ ]:
```
#replacing the missing values with the most frequent value
df['bmi'].fillna(df['bmi'].mode()[0], inplace=True)
```

Check values and their count in the columns

In [ ]:
```
print(df['ever_married'].value_counts())
print(df['work_type'].value_counts())
print(df['gender'].value_counts())
print(df['Residence_type'].value_counts())
print(df['smoking_status'].value_counts())
```

```
ever_married
Yes     3353
No      1757
Name: count, dtype: int64
work_type
Private          2925
Self-employed     819
children          687
Govt_job          657
Never_worked       22
Name: count, dtype: int64
gender
Female    2994
Male      2115
Other        1
Name: count, dtype: int64
Residence_type
Urban    2596
Rural    2514
Name: count, dtype: int64
smoking_status
never smoked       1892
Unknown            1544
formerly smoked     885
smokes              789
Name: count, dtype: int64
```

Replacing the values in columns with numerical values

- Residence Type: Urban = 1, Rural = 0
- Smoking Status: formerly smoked = 1, never smoked = 2, smokes = 3, Unknown = 0
- Ever_Maried : Yes = 1, No = 0
- Gender : Male = 1, Female = 0, Other = 2

- Work Type : Private = 0, Self-employed = 1, children = 2, Govt_job = 3, Never_worked = 4

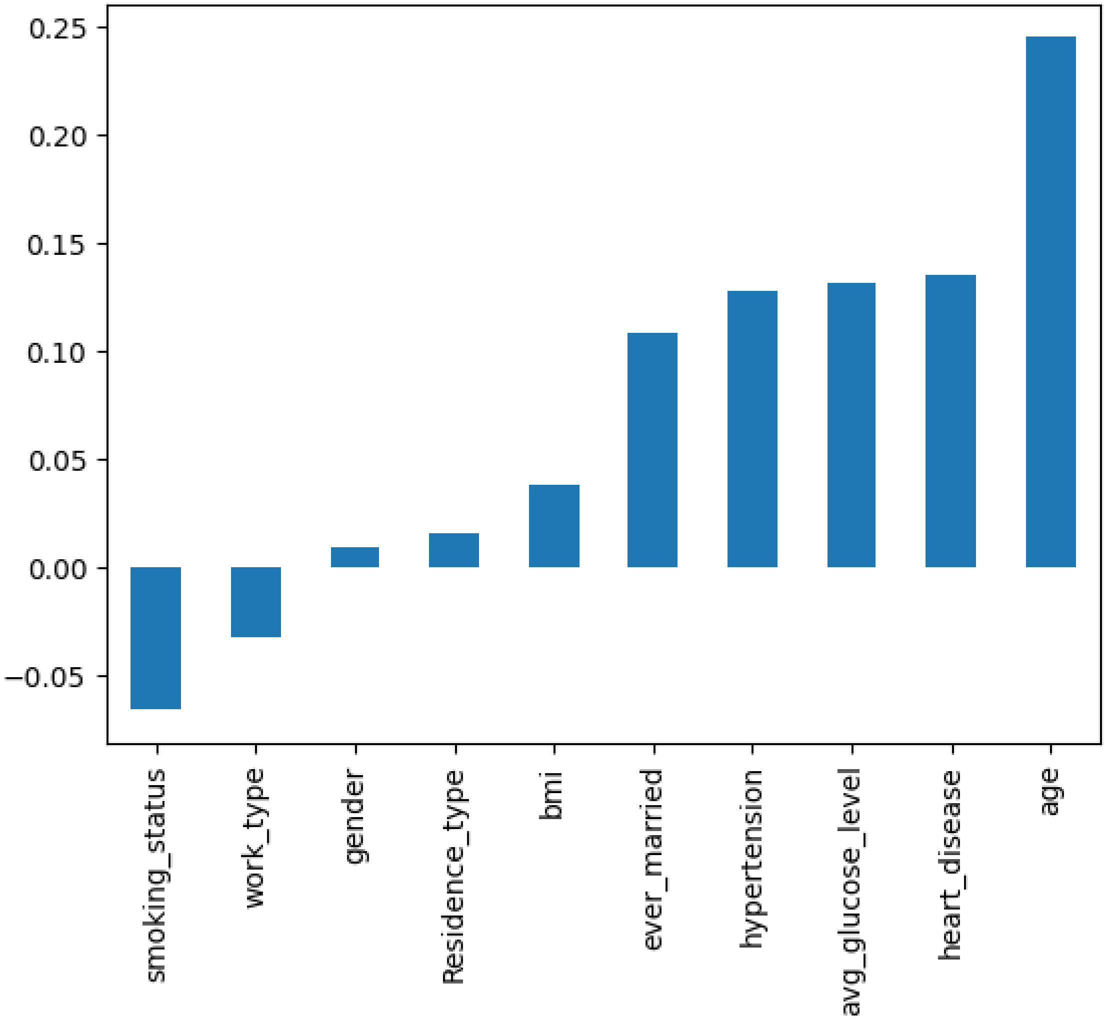
In [ ]:
```
df['ever_married'].replace({'Yes':1, 'No':0}, inplace=True)
df['gender'].replace({'Male':1, 'Female':0,'Other':2}, inplace=True)
df['Residence_type'].replace({'Urban':1, 'Rural':0}, inplace=True)
df['smoking_status'].replace({'formerly smoked':0, 'never smoked':1, 'smokes':2,
df['work_type'].replace({'Private':0, 'Self-employed':1, 'children':2, 'Govt_jot
```

Exploratory Data Analysis

Find correlation between the variables

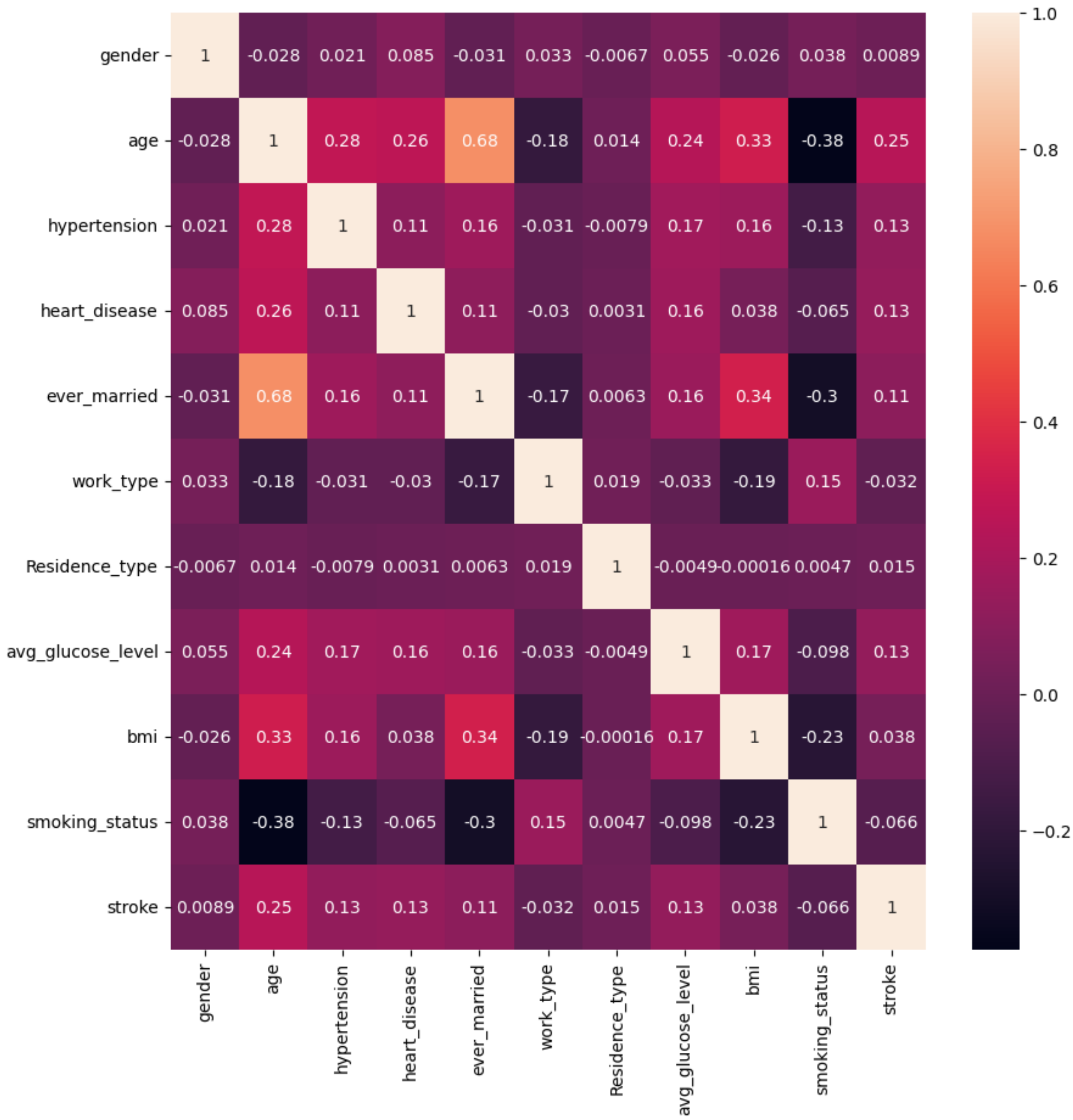
In [ ]:
```
df.corr()['stroke'][:-1].sort_values().plot(kind='bar')
```

Out[ ]: <Axes: >



In [ ]:
```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(), annot=True)
```

Out[ ]: <Axes: >



In [ ]:
```python
# replace age with number wrt to age group
# 0 = 0-12 , 1 = 13-19 , 2 = 20-30 , 3 = 31-60 , 4 = 61-100
df['age'] = pd.cut(x=df['age'], bins=[0, 12, 19, 30, 60, 100], labels=[0, 1, 2,
df.head()
```

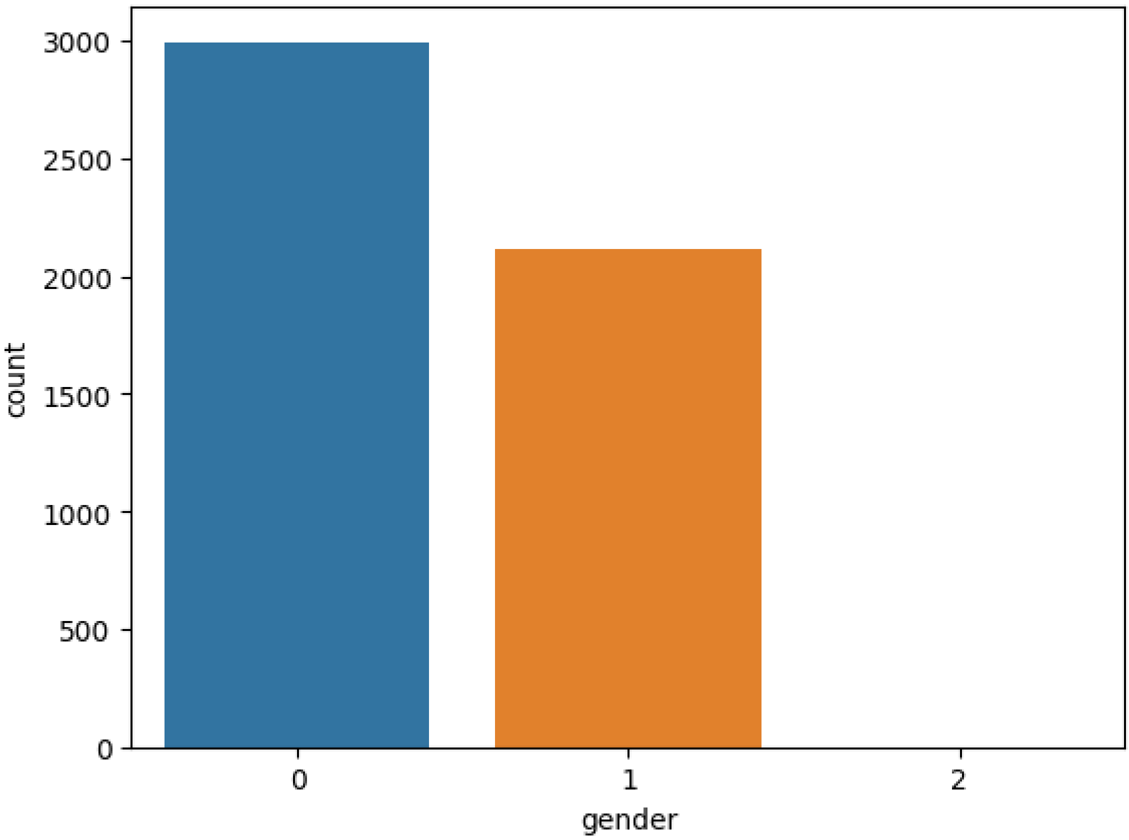
Out[ ]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type
0	1	4	0	1	1	0	1
1	0	4	0	0	1	1	0
2	1	4	0	1	1	0	0
3	0	3	0	0	1	0	1
4	0	4	1	0	1	1	0

Visulaizing the data

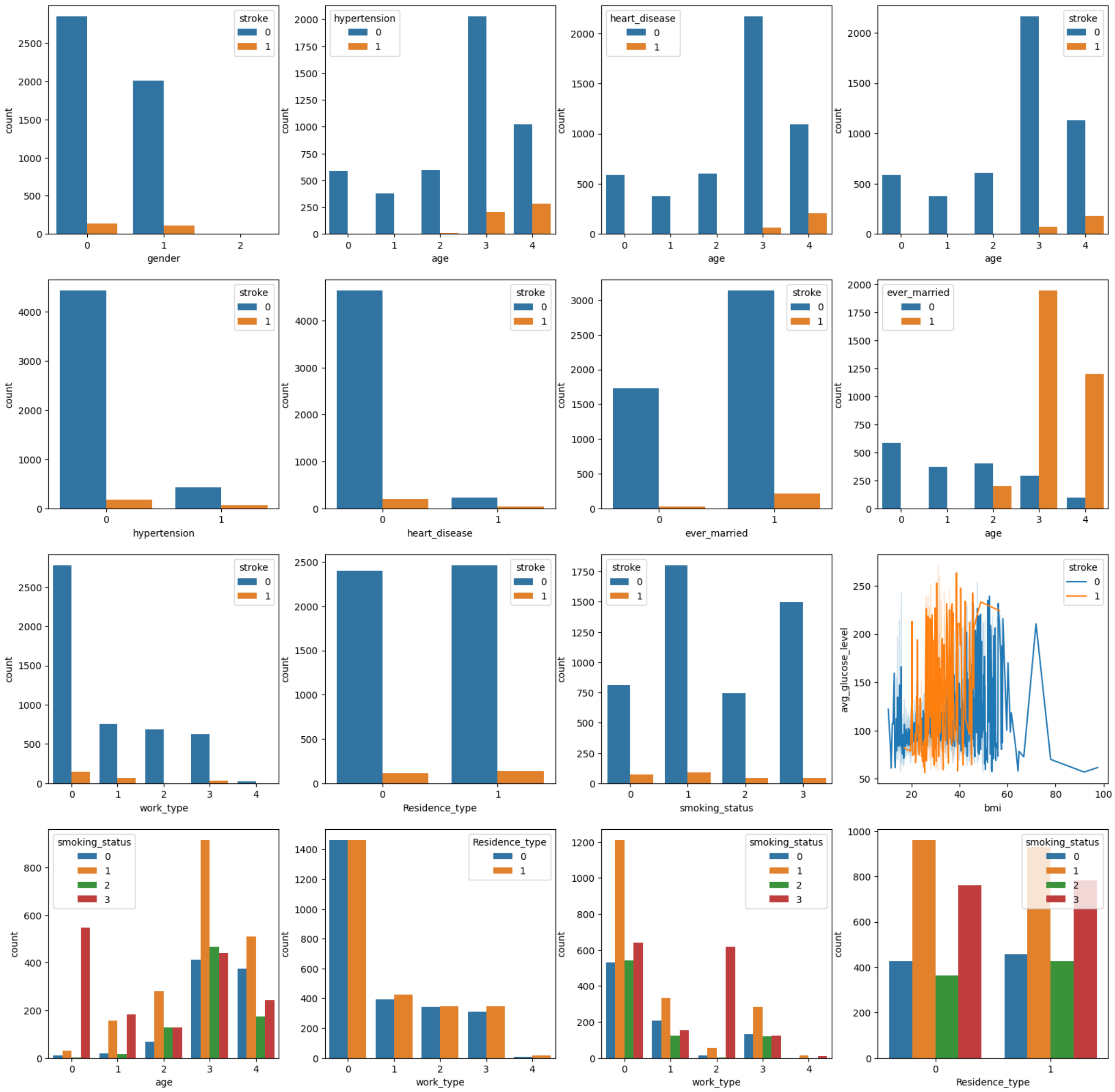
In [ ]:
```python
sns.countplot(x = 'gender', data = df)
```

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



In [ ]:
```
fig, ax = plt.subplots(4,4,figsize=(20, 20))
sns.countplot(x = 'gender', data = df,hue = 'stroke', ax=ax[0,0])
sns.countplot(x = 'age', data = df,hue = 'hypertension', ax=ax[0,1])
sns.countplot(x = 'age', data = df,hue = 'heart_disease', ax=ax[0,2])
sns.countplot(x = 'age', data = df,hue = 'stroke', ax=ax[0,3])
sns.countplot(x = 'hypertension', data = df,hue = 'stroke', ax=ax[1,0])
sns.countplot(x = 'heart_disease', data = df,hue = 'stroke', ax=ax[1,1])
sns.countplot(x = 'ever_married', data = df,hue = 'stroke', ax=ax[1,2])
sns.countplot(x = 'age', data = df,hue = 'ever_married', ax=ax[1,3])
sns.countplot(x = 'work_type', data = df,hue = 'stroke', ax=ax[2,0])
sns.countplot(x = 'Residence_type', data = df,hue = 'stroke', ax=ax[2,1])
sns.countplot(x = 'smoking_status', data = df,hue = 'stroke', ax=ax[2,2])
sns.lineplot(x = 'bmi', y = 'avg_glucose_level', data = df,hue = 'stroke', ax=ax
sns.countplot(x = 'age', data = df,hue = 'smoking_status', ax=ax[3,0])
sns.countplot( x = 'work_type', data = df,hue = 'Residence_type', ax=ax[3,1])
sns.countplot(x = 'work_type', data = df,hue = 'smoking_status', ax=ax[3,2])
sns.countplot(x = 'Residence_type', data = df,hue = 'smoking_status', ax=ax[3,3]
```

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



Train Test Split

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(df.drop('stroke', axis=1), c
```

Model Training

Logistic Regression

In [ ]:
```
lr = LogisticRegression()
lr
```

Out[ ]:
```
▾ LogisticRegression
LogisticRegression()
```

In [ ]:
```
#training the model
lr.fit(X_train, y_train)
lr.score(X_test, y_test)
```

```
C:\Users\DELL\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2k
fra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\linear_model\_lo
gistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[ ]: 0.9393346379647749

In [ ]:
```
#testing the model
lr_pred = lr.predict(X_test)
accuracy_score(y_test, lr_pred)
```

Out[ ]: 0.9393346379647749

Support Vector Machine (SVM)

In [ ]:
```
from sklearn.svm import SVC
svm = SVC()
svm
```

Out[ ]:
```
▾ SVC
SVC()
```

In [ ]:
```
#training the model
svm.fit(X_train, y_train)
svm.score(X_test, y_test)
```

Out[ ]: 0.9393346379647749

In [ ]:
```
#testing the model
sv_pred = svm.predict(X_test)
accuracy_score(y_test, sv_pred)
```

Out[ ]: 0.9393346379647749

Decision Tree Classifier

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt
```

Out[ ]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [ ]:
```
#training the model
dt.fit(X_train, y_train)
dt.score(X_test, y_test)
```

Out[ ]: 0.9099804305283757

In [ ]:
```
#testing the model
dt_pred = dt.predict(X_test)
accuracy_score(y_test, dt_pred)
```

Out[ ]: 0.9099804305283757

K-Nearest Neighbors (KNN)

In [ ]:
```
knn = KNeighborsClassifier()
knn
```

Out[ ]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [ ]:
```
#training the model
knn.fit(X_train, y_train)
knn.score(X_test, y_test)
```

Out[ ]: 0.9373776908023483

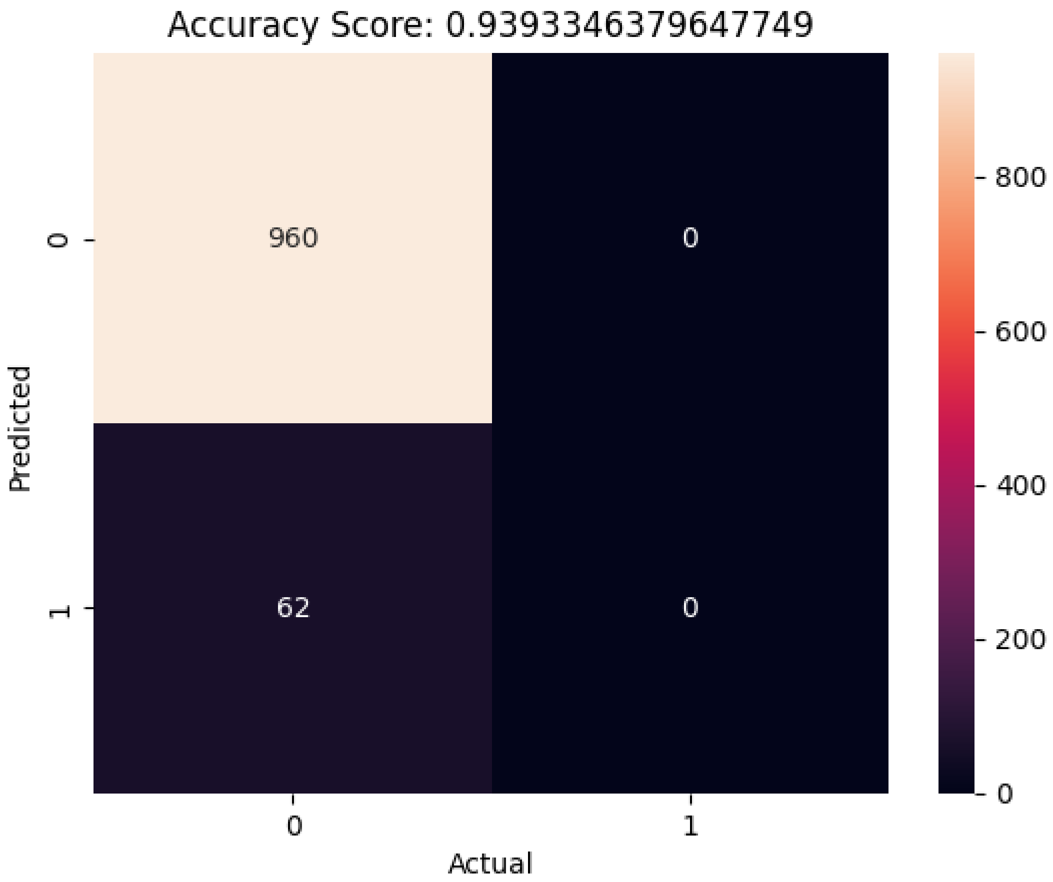
In [ ]:
```
#testing the model
knn_pred = knn.predict(X_test)
accuracy_score(y_test, knn_pred)
```

Out[ ]: 0.9373776908023483

Model Evaluation

Logistic Regression

In [ ]:
```
sns.heatmap(metrics.confusion_matrix(y_test, lr_pred), annot=True, fmt='d')
plt.title('Accuracy Score: {}'.format(accuracy_score(y_test, lr_pred)))
plt.ylabel('Predicted')
plt.xlabel('Actual')
plt.show()
```

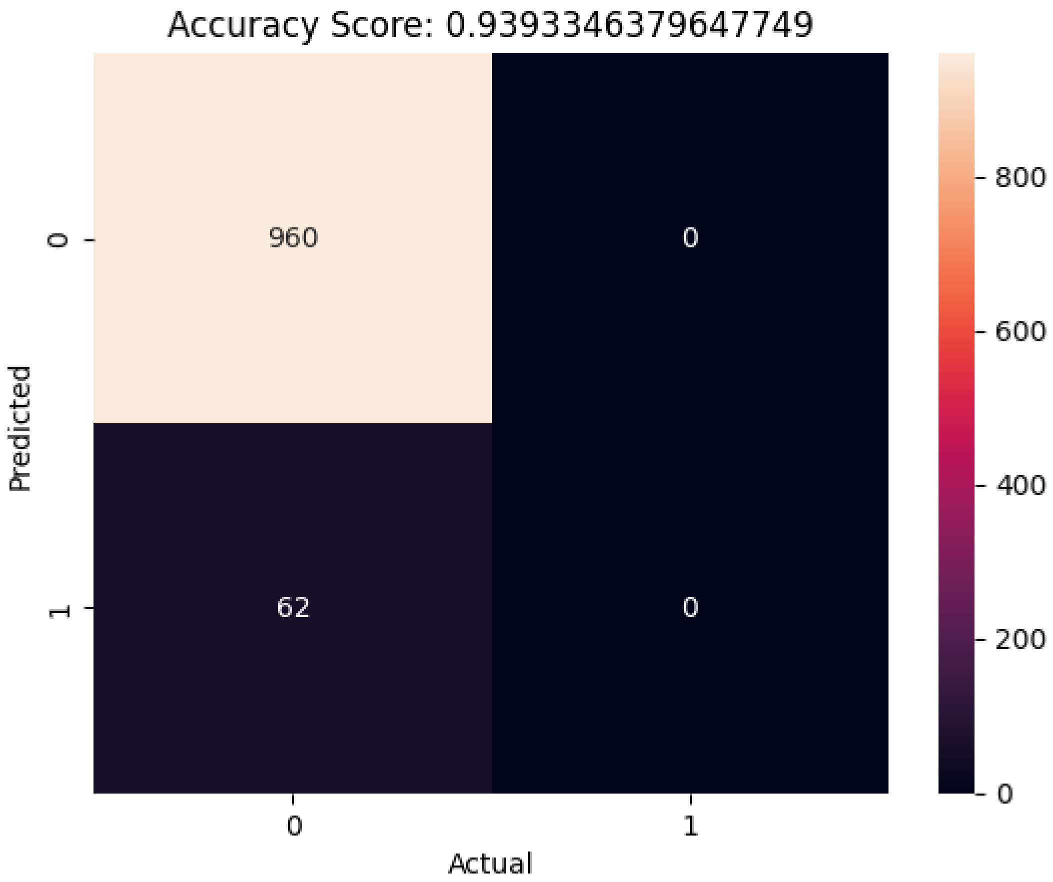


In [ ]:
```
print('Logistic Regression Model Accuracy Score:',accuracy_score(y_test, lr_pred
print('Logistic Regression Model F1 score: ',metrics.f1_score(y_test, lr_pred))
print('Logistic Regression Model Mean Absolute Error: ',metrics.mean_absolute_er
print('Logistic Regression Model Mean Squared Error: ',metrics.mean_squared_erro
print('Logistic Regression Model log loss: ',log_loss(y_test, lr_pred))
```

```
Logistic Regression Model Accuracy Score: 0.9393346379647749
Logistic Regression Model F1 score:  0.0
Logistic Regression Model Mean Absolute Error:  0.060665362035225046
Logistic Regression Model Mean Squared Error:  0.060665362035225046
Logistic Regression Model log loss:  2.1866012819229583
```

Support Vector Machine (SVM)

In [ ]:
```
sns.heatmap(metrics.confusion_matrix(y_test, sv_pred), annot=True, fmt='d')
plt.title('Accuracy Score: {}'.format(accuracy_score(y_test, sv_pred)))
plt.ylabel('Predicted')
plt.xlabel('Actual')
plt.show()
```

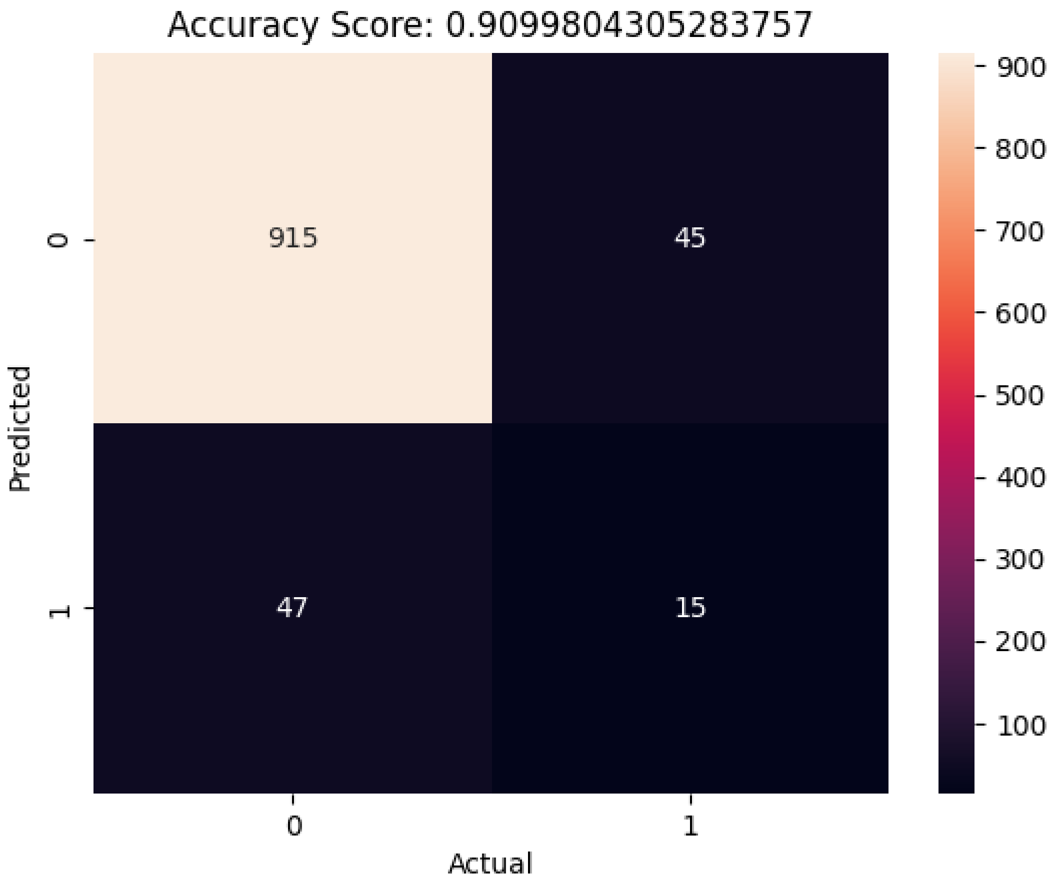



```
In [ ]: print('SVM Model Accuracy Score:',accuracy_score(y_test, sv_pred))
        print('SVM Model F1 score: ',metrics.f1_score(y_test, sv_pred))
        print('SVM Model Mean Absolute Error: ',metrics.mean_absolute_error(y_test, sv_p
        print('SVM Model Mean Squared Error: ',metrics.mean_squared_error(y_test, sv_pre
        print('SVM Model log loss: ',log_loss(y_test, sv_pred))
```

```
SVM Model Accuracy Score: 0.9393346379647749
SVM Model F1 score:  0.0
SVM Model Mean Absolute Error:  0.060665362035225046
SVM Model Mean Squared Error:  0.060665362035225046
SVM Model log loss:  2.1866012819229583
```

Decision Tree Classifier

```
In [ ]: sns.heatmap(metrics.confusion_matrix(y_test, dt_pred), annot=True, fmt='d')
        plt.title('Accuracy Score: {}'.format(accuracy_score(y_test, dt_pred)))
        plt.ylabel('Predicted')
        plt.xlabel('Actual')
        plt.show()
```

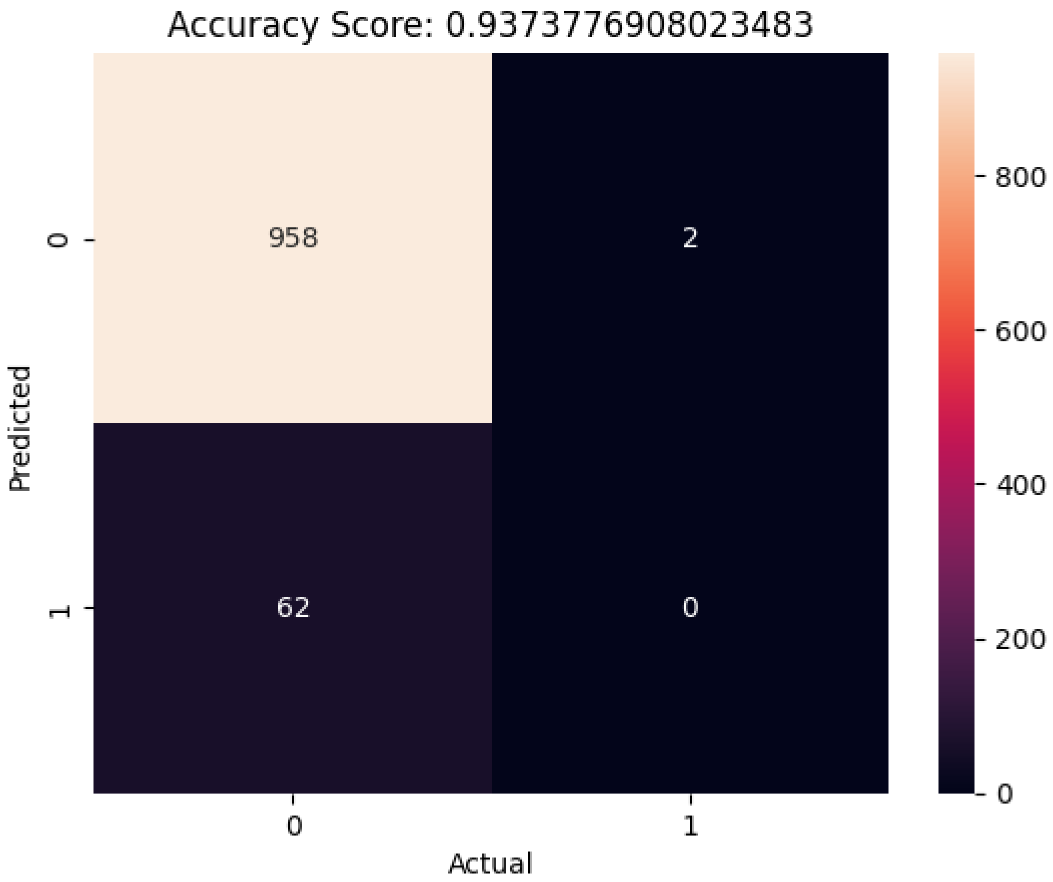


In [ ]:
```
print('Decision Tree Model Accuracy Score:',accuracy_score(y_test, dt_pred))
print('Decision Tree Model F1 score: ',metrics.f1_score(y_test, dt_pred))
print('Decision Tree Model Mean Absolute Error: ',metrics.mean_absolute_error(y_
print('Decision Tree Model Mean Squared Error: ',metrics.mean_squared_error(y_te
print('Decision Tree Model log loss: ',log_loss(y_test, dt_pred))
```

```
Decision Tree Model Accuracy Score: 0.9099804305283757
Decision Tree Model F1 score:  0.2459016393442623
Decision Tree Model Mean Absolute Error:  0.09001956947162426
Decision Tree Model Mean Squared Error:  0.09001956947162426
Decision Tree Model log loss:  3.2446341602727773
```

K-Nearest Neighbors (KNN)

In [ ]:
```
sns.heatmap(metrics.confusion_matrix(y_test, knn_pred), annot=True, fmt='d')
plt.title('Accuracy Score: {}'.format(accuracy_score(y_test, knn_pred)))
plt.ylabel('Predicted')
plt.xlabel('Actual')
plt.show()
```

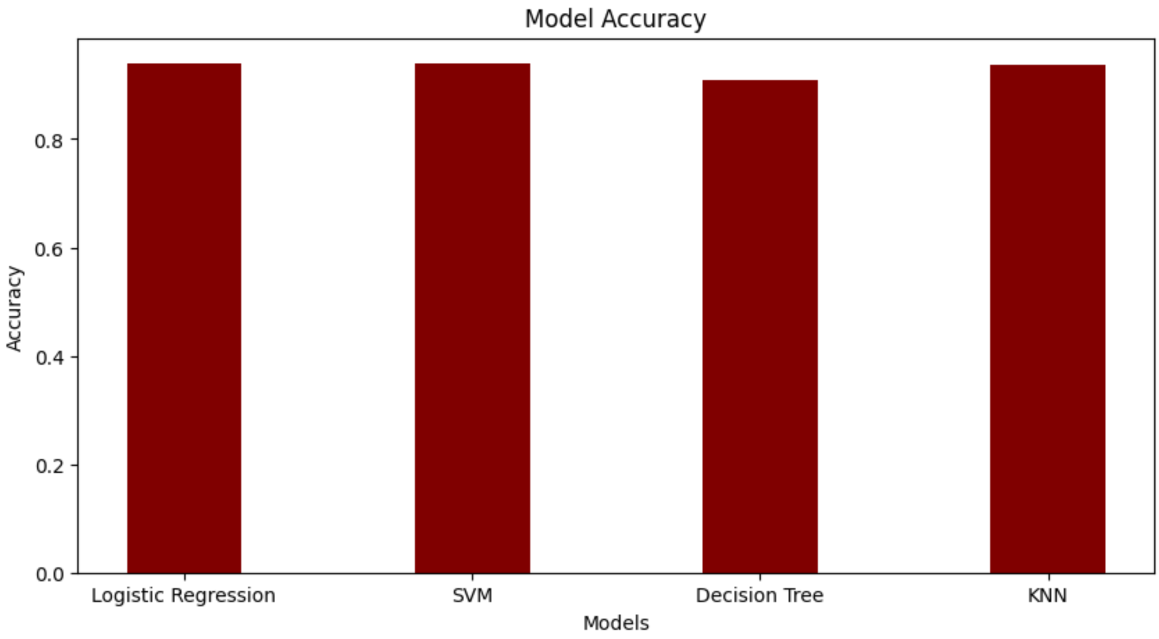



```
In [ ]: print('KNN Model Accuracy Score:',accuracy_score(y_test, knn_pred))
print('KNN Model F1 score: ',metrics.f1_score(y_test, knn_pred))
print('KNN Model Mean Absolute Error: ',metrics.mean_absolute_error(y_test, knn_
print('KNN Model Mean Squared Error: ',metrics.mean_squared_error(y_test, knn_pr
print('KNN Model log loss: ',log_loss(y_test, knn_pred))
```

```
KNN Model Accuracy Score: 0.9373776908023483
KNN Model F1 score:  0.0
KNN Model Mean Absolute Error:  0.06262230919765166
KNN Model Mean Squared Error:  0.06262230919765166
KNN Model log loss:  2.2571368071462796
```

Model Comparison

```
In [ ]: models = ['Logistic Regression', 'SVM', 'Decision Tree', 'KNN']
accuracy = [accuracy_score(y_test, lr_pred), accuracy_score(y_test, sv_pred), ac
plt.figure(figsize=(10,5))
plt.bar(models, accuracy, color = 'Maroon', width = 0.4)
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')
plt.show()
```



Conclusion

The model accuracies of Logistic Regression, SVM and KNN are quite similar i.e. 93.8 %. The accuracy of Decision Tree Classifier is 91.8 %. So, we can use any of these models to predict the heart stroke.

According to the graphs age v/s hypertension, heart disease showing chances of stroke, the number of person having a stroke shows dependece upon heart disease and hypertension. But when we plot the graph of heart disease and hypertension against the stroke, the persons with lower chances of hypertension and heart disease has increased chances of stroke. This is a peculiar thing and needs to be investigated further. In addition to that non somkers have higher chances of stroke than smokers. This is also a peculiar thing and needs to be investigated further. However person having BMI between20 to 50 have higher chances of stroke.

Last but not least other features such as martial status, residence type as well as work type are showing effect on the chances of stroke.